Multiparty Private Set Intersection Cardinality and Its Applications

Jiahui Gao* Ni Trieu* Avishay Yanai†

September 10, 2023

Abstract

We describe a new paradigm for multi-party private set intersection cardinality (PSI-CA) that allows n parties to compute the intersection size of their datasets without revealing any additional information. We explore a variety of instantiations of this paradigm. Our protocols avoid computationally expensive public-key operations and are secure in the presence of a semi-honest adversary.

We demonstrate the practicality of our PSI-CA with an implementation. For $n = 16$ parties with data-sets of 2^{20} items each, our server-aided variant takes 71 seconds. Interestingly, in the server-less setting, the same task takes only 7 seconds. To the best of our knowledge, this is the first 'special purpose' implementation of a multi-party PSI-CA from symmetric-key techniques (i.e., an implementation that does not rely on a generic underlying MPC).

We study two interesting applications – heatmap computation and associated rule learning (ARL) – that can be computed securely using a dot-product as a building block. We analyse the performance of securely computing heatmap and ARL using our protocol and compare that to the state-of-the-art.

1 Introduction

Secure multi-party computation (MPC) allows a set of parties to jointly invoke a distributed computation while ensuring correctness, privacy, and more. Garbled circuit [49, 23, 6] is a popular generic technique for secure computation, which has been enjoyed notable optimizations in recent years (e.g. [41]). However, for concrete applications, special-purpose protocols significantly improve performance compared to circuit-based approaches.

In this work, we study Private Set Intersection Cardinality (PSI-CA), a special case of MPC, that allows multiple parties to compute the intersection size of their private sets without revealing additional information. PSI itself has been motivated by many real-world applications such as contact discovery [27]. Over the last several years PSI has become truly practical with extremely fast cryptographically secure implementations [12, 40, 37, 22]. In the setting of two parties, PSI with post-processing (a.k.a circuit-based PSI), especially PSI-CA, has recently drawn more attention with several applications, such as measuring the effectiveness of online advertising [26], limiting the spread of Child Sexual Abuse Material (CSAM) [8], and private contact tracing related to COVID-19 [7, 18, 15]. However, the state-of-the-art PSI-CA is only efficient in the two-party setting [26, 18, 15]. This work considers a natural generalization to the multi-party setting, which

*Arizona State University, {jhgao, nitrieu}@asu.edu

†VMware Research, ay.yanay@gmail.com

opens the opportunity for richer applications, like the two we showcase below. The state-of-the-art protocol for PSI-CA in the multi-party setting [11] relies on secret-shared computation [13], which might not scale well for a large number of parties. In this work we present a scalable protocol for PSI-CA in the multi-party setting.

Moreover, we present a new protocol, called DotProd, where n parties may compute a sum of element-wise products of their binary vectors without revealing any additional information. Mathematically, suppose party P_i holds the m-element vector x_i, then the parties obtain $\sum_{j=1}^{m} \prod_{i=1}^{n} x_i[j]$, where $x_i[j]$ is the j^{th} element of the vector x_i. Note that in the two-party case, the computation is exactly of the dot product $x_1 \cdot x_2$. We demonstrate the efficiency of our protocols through two real-world applications: a COVID-19 heatmap computation based on PSI-CA and an associated rule learning (ARL) based on DotProd.

In the rest of this section, we will present the related work of PSI-CA and its applications. Additionally, we will delve into the technical overview and outcomes of our proposed protocols. To establish a foundation, Section 2 presents the necessary preliminaries. Furthermore, we will introduce two novel cryptographic gadgets, namely Server-Aided Shuffled OPRF and Server-Aided OPPRF, in Section 3. These gadgets serve as the fundamental building blocks for our PSI-CA protocols, which will be discussed in Section 4. The practical applications of PSI-CA will be explored in Section 5. Lastly, in Section 6, we will evaluate the performance of our PSI-CA protocols and provide a comparison with existing approaches.

1.1 State-of-the-Art for PSI Cardinality

Private Set Intersection Cardinality (PSI-CA) is a variant of PSI in which the parties learn the intersection size and nothing else. In this work, we also focus on *server-aided* PSI-CA constructions. By "server-aided", we refer to cases where the parties perform PSI-CA computation with the help of semi-honest cloud server(s). To the best of our knowledge, this work proposes the first special-purpose PSI-CA protocols from symmetric-key techniques that work in the multi-party setting.

We start with discussing PSI-CA works in the two-party setting. Clearly, one can use circuit-based PSI [38] to implement PSI-CA. However, this generic solution is expensive due to the secure computation inside the circuit. For the special-purpose two-party PSI-CA constructions, the work [26] extends the classic DH-based PSI protocol [34] to support two-party PSI-CA by having a sender shuffle the PRFs of their items before returning to the receiver. Epione [46] also proposed a protocol that is suitable to the unbalanced, client-server setting, in which the server has a large database of m_1 items and the client has a small database of m_2 items. The protocol, however, requires $O(m_1 + m_2)$ expensive public-key operations (group exponentiation). Delegated PSI-CA[18] improves the efficiency of the two-party PSI-CA protocol on the client's device, Catalic [18] proposes a delegated system in which the client (i.e. PSI-CA receiver) can shift most of its PSI-CA computation to multiple untrusted servers while preserving privacy. However, Catalic system requires at least two non-colluding cloud servers with a heavy computation/communication cost. Based on oblivious switching network (OSN), [21] proposes a two-party PSI-CA (so-called OSN-based PSI-CA) which is better than circuit-based PSI-CA protocol [38] in terms of communication cost and running time in the WAN setting. However, it has both communication and computation complexity $O(m \log(m))$ for a set size m due to the expensive OSN construction. Dittmer et al. [15] introduces a variant of two-party PSI-CA (so-called weighted PSI-CA) in which each token of the client has an associated secret weight. The weighted PSI-CA is based on cheap Function Secret Sharing (FSS) constructions [9, 10], thus it is efficient on both client's and server's sides. However, their construction assumes that there exist two non-colluding servers, each holding an identical input set.

A multi-party PSI-CA protocol was first proposed by Kissner and Song [30]. The protocol of [30] is based on oblivious polynomial evaluation which is implemented using additively homomorphic encryption. The basic idea is to represent a dataset as a polynomial whose roots are its elements, and send the homomorphic encryptions of the coefficients to other parties so that they can evaluate the encrypted polynomial on their inputs. The protocol of [30] has a quadratic computation and communication complexity in both the size of dataset and the number of parties.

Mohassel et al. [36] proposed a PSI-CA protocol, but on secret shared data in the honest-majority three-party setting, which is different than the setting in this paper, as we consider a setting with any number of parties, in which the input does not have to be in a secret-sharing form. However, one can extend the protocol of [36] to support the multi-party PSI-CA where all the parties secret-share their input to the three parties of [36] which then jointly compute the final output. We discuss the extension and compare the performance of our protocol and [36]'s in Section 6.2.

Chandran et al. [11] proposed an efficient PSI (not PSI-CA), which can be extended to circuit-based PSI. Hence, one could combine their extended protocol with a circuit that computes the size of the intersection to obtain a protocol for PSI-CA. At the technical core, [11] is built on n-party secret-sharing functionalities introduced by [13]. Their use of generic secure computation protocol for a specific problem (of PSI-CA) makes their extended protocol less attractive. We also compare the performance of our protocols and [11] in Section 6.2.

Very recently, Fenske et al. [19] proposed an efficient multi-party PSI-CA procotol in the outsourcing setting. Their approach makes use of K servers, with the assumption that at least one of the servers is not colluding with other participants. When $K = 1$, their protocol is comparable to our server-aided one, as both require a non-colluding server. However, our protocol outperforms theirs in this scenario, as we employ symmetric-key operations while [19] heavily relies on the additiviely homomorphic encryption. For instance, in the case of $n = 8$ and $m = 2^{16}$, our protocol can compute PSI-CA within 3 seconds. In contrast, the protocol [19] necessitates approximately 2 hours for $n = 5$ and $m = 30000$, as indicated in their Figure 10. This demonstrates the efficiency and superiority of our approach in terms of computational time. It is also worth noting that the protocol [19] only works in the server-aided setting, whereas in this work we also propose a way to work without such an entity, which we call the "server-less" setting.

1.2 Secure Dot Product and Its Applications

Dot-product plays a key role in machine learning and data analysis tasks. Its implementation in a privacy-preserving setting remains expensive as it requires either generating Beaver triples [5] or using fully homomorphic encryption (FHE). There is a long list of results for secure computation of dot product or linear algebra in general [1, 25, 45, 51, 50, 14, 4]. For the applications that we consider in this paper, namely, Covid-heatmap and ARL, dot-product of *sparse vectors* would be sufficient. Many algorithms for linear algebra operations, like matrix multiplication, leverage an apriori knowledge of the operands being sparse, and sometimes these algorithms can even be computed securely, without degrading their asymptotic complexity. None of the above works, however, address the problem of dot product in a setting where the vectors are sparse. The most relevant works to ours are [47, 48, 16, 4, 44].

To the best of our knowledge, Vaidya and Clifton [47] were the first to study secure computation of scalar product of two m-element vectors in the two-party setting and its application to privacy-preserving association rule learning (ARL). Their dot product protocol heavily relies on public-key operations, and requires four communication rounds, communication complexity of $O(m)$ and computation complexity of $O(m^2)$.

Their follow-up work [48] is based on PSI, which makes the complexity dependent only of t, where t is the upper-bound on the Hamming weight of the vectors. They also propose a protocol for the multi-party setting, which requires a commutative one-way hash function so that the input from each party can be encrypted by a common set of keys. The resulting ciphertexts are the same if the original values are the same. Although efficient, their protocol introduces an undesirable leakage; specifically, it leaks the items in the intersection (rather only their sum). Moreover, their protocol is insecure when the input domain is relatively small (e.g. of size 2^{30}) as one party could easily perform a brute force attack [39]. To handle the latter security issue, [16] studied a two-party ARL and proposed a solution via PSI that is built on the Goldwasser-Micali Encryption [24] and Oblivious Bloom Intersection [17]. Their protocol still leaks the items in the intersection, and became much more expensive than the protocol we present in this paper. In addition, they did not consider an extension to the multi-party case.

Recently, Bampoulidis et al. [4] studies COVID-19 heatmap computation and proposes secure dot product based on homomorphic encryption with several optimizations. However, the number of required HE operations is $O(m)$ (regardless of the Hamming weight of the vectors), which makes their protocol expensive. Schoppmann et al. [44] presents efficient two-party protocols for several common sparse linear algebra operations including sparse matrix-vector multiplication. The main building block of their protocols is a new functionality – Read-Only Oblivious Map (ROOM). Using ROOM, the cost of the secure matrix-vector multiplication is dependent only on the number of non-zero entries, instead of the operands' size. However, in all three ROOM constructions the parties invoke generic secure computation in order to obtain a secret-shared output. We compare the performance of our protocol to a ROOM-based dot-product in Section 6.1.

1.3 Our Results and Techniques

1.3.1 Our PSI-CA Approach:

We present a new multi-party PSI-CA protocol paradigm with an assumption that a subset of particular parties does not collude. We offer two variants of our protocol. The first protocol relies on a non-colluding semi-honest server that has no input. It is optimized for the number of communication rounds between parties; that is, the protocol leverages a star network topology, where parties mostly communicate with the server. The second protocol removes the need of a server by reducing the problem of n-party PSI-CA to the problem of server-aided $(n-1)$-party PSI-CA with use of a semi-honest party P_n *who may have an input*. The base case with $n = 2$ can be instantiated efficiently by two-party server-aided PSI protocol of Kamara et al. [28]. However, [28] is only for PSI itself (not PSI-CA)[1]. We simplify their PSI protocol and present a new server-aided two-party PSI-CA in Section 4.1.

The main building blocks of our multiparty PSI-CA protocols are oblivious key-value store (OKVS) data structure [22], and/or Oblivious Programmable PRF (OPPRF) [31]. To this end, we propose a very simple and efficient protocol for server-aided OPPRF, which we believe to be

[1] Note that [28] has a protocol for multiparty PSI, but it reveals intersection items of each pair-wise parties sets to the server and is non-trivial to support PSI-CA.

of independent interest. Our server-aided OPPRF is based on a two-party server-aided shuffled OPRF, a functionality we formally define in Section 3.1.

We provide an implementation of server-aided and server-less variants of our PSI-CA approach for $n > 2$. To the best of our knowledge, this is the first 'special-purpose' implementation of multi-party PSI-CA from symmetric-key techniques that does not rely on generic secure computation. We find that multi-party PSI-CA is practical, by evaluating our protocols over settings with million items sets and 16 parties. The main reason for the efficiency of our protocol is its reliance on fast symmetric-key primitives. This is in contrast with prior multi-party PSI-CA protocols, which require expensive public-key operations for each item [30] or computation on secret-shared data [11].

Interestingly, the server-less PSI-CA variant is about 10× faster than the server-aided one. We consider colluding model in the semi-honest setting which is introduced in detail in Section 2.1. The two variants, however, offer different security guarantees. Specifically, the former is secure in the presence of an adversary who may passively corrupt any subset from $\{P_3, \ldots, P_n\}$ or one of P_1,P_2 or P_n (i.e. P_1, P_2 and P_n are non-colluding). The latter (server-aided PSI-CA) is secure in the presence of an adversary who may passively corrupt any strict subset of $\{P_1, P_3, \ldots, P_n\}$ or $\{P_2, P_3, \ldots, P_n\}$ (i.e. P_1 and P_2 do not collude) or passively corrupt the cloud server $\mathcal{C}$. In some sense, one may look at the server-less variant as a multi-server-aided PSI-CA but the servers have their private input. Hence, we can use our efficient server-aided OPPRF (instead of the two-party OPPRF [31]) in the server-less PSI-CA protocol, which may explain why it is possible to get a better performance in this case. In the server-less variant, we assign the non-colluding party P_1 the role of a server in the server-aided OPPRF protocol.

The security model employed in this work deviates from the commonly known concept of "threshold security". Rather, we adopt a specific but sufficiently general access structure, in which a designated subset of parties does not collude. Although this approach differs from the conventional notion of threshold security, we do believe our approach can be used as a stepping stone toward achieving security in the 'standard' threshold access structure.

Note that in practice, a server-aided model can be reasonable. Performance is critical and often it makes sense given that the alternative has a weaker security guarantee. For example, in the federated learning setting, there is a server and many clients where the server helps training a machine learning model for the benefit of the clients. In this work, we motivate our protocols with two real-world applications in which using a non-colluding, but semi-honest server, makes complete sense. For example, in the Covid-19 heatmap computation, an established company (e.g. Google or Apple) can play the role of the server.

1.3.2 Our Multi-party Dot-Product of Binary Vectors (DotProd):

We propose a new protocol for computing the sum of element-wise products of n sparse binary vectors (so-called multiple dot product, DotProd). Let us begin with the simpler case, where $n = 2$, known as secure dot product. One would expect a solution for a dot product of m-elements vectors to incur communication overhead of at least $O(m)$, for the very fact that the parties need to first input those elements (which usually involves some sort of encryption or secret sharing on each element). In this work, we show that the communication and computation complexity is independent of m and can be reduced to $O(t)$, where t is the upper bound on the Hamming weight of the vectors. This improvement is significant when the vectors are sparse (i.e. $t = o(m)$).

For an m-element binary vector x we define $\mathbf{idx}(x) = \{i \in [m] \mid x[i] = 1\}$ to be the set of non-zero indices in x. Suppose the receiver P_0 and the sender P_1 hold an m-element binary sparse vector x_0 and x_1, respectively. The vectors are sparse and have the number of non-empty elements bounded by $t = o(m)$. As a very simple warm-up, we consider a non-secure dot product computation with the communication complexity cost of $O(t)$. Given the input vector x_0, the receiver computes $A_0 = \mathbf{idx}(x_0)$ and the sender computes $A_1 = \mathbf{idx}(x_1)$. The sender then sends A_0 to the receiver, who is able to compute the dot product $x \cdot y$ by computing the intersection $A_0 \cap A_1$ and outputting its cardinality $|A_0 \cap A_1|$.

The main advantage of the above solution is to reduce dependency on the length of the vectors, especially when the input vectors are sparse. To compute $x_0 \cdot x_1$ securely, the parties run a private set intersection cardinality protocol (PSI-CA) where P_0 inputs A_0 and P_1 inputs A_1. This idea, however, has received little attention due to the large overhead required to compute PSI-CA. In this work, we propose a lightweight server-aided PSI-CA construction to improve the performance of the secure dot product. Our two-party protocol relies on only PRF. As a result, our protocols are simple and efficient, with a communication and computation complexity $O(t)$, so is our secure dot product DotProd.

We then extend DotProd to the multi-party case. Given an input vector x_i, party P_i computes $A_i = \mathbf{idx}(x_i)$. It is easy to see that the sum of element-wise products of the vectors is equal to the size of their intersection, namely, $\sum_{j=1}^{m} \prod_{i=1}^{n} x_i[j] = |\bigcap_{i=1}^{n} A_i|$. We implement the multi-party DotProd using our multi-party PSI-CA.

1.3.3 Application to PSI-CA and DotProd:

We show that our PSI-CA and DotProd techniques can be used to implement and improve the performance of several privacy-preserving applications. More specifically, we consider two running examples: COVID-19 heatmap computation and associated rule learning (ARL).

In the COVID-19 heatmap problem, we consider a scenario where the Department of Health and Human Services (HHS) wants to learn areas with a higher chance of getting infected with the disease without knowing the travel route of infected individuals. The heatmap can be implemented by computing the vector-matrix multiplication as $x^\top Y$, where x and Y are as follows: x is a binary vector of size N, held by the HHS, such that $x[i] = 1$ if the ith user has tested positive to COVID-19 and $x[i] = 0$ otherwise; and $Y = (y_1, \ldots, y_m) \in \mathbb{Z}_2^{N \times m}$ is a user-location matrix, held by a network operator, such that the ith element of the column vector y_j indicates whether the ith user has recently visited the jth location. In that case $y_j[i] = 1$ and otherwise $y_j[i] = 0$. Clearly, $z = x^\top Y$ is an m-element vector where the ith element is equal to the number of users who have tested positive and recently visited the ith location. [4] proposes different optimizations on HE to implement a secure dot product, which still requires $O(Nm)$ independent multiplications (regardless of the Hamming weight of the vectors). In the heatmap example above, we observe that the vector x is sparse because the proportion of diagnosed individuals per day among all N subscribed individuals is small (e.g, 0.01-1% would be a large percentage [2]). Similarly, the matrix Y is also sparse due to people's localized travel habits. In Section 5.2, we apply our DotProd protocol to compute COVID-19 heatmap. In addition, [4] only supports a two-party computation between the HHS and a network provider. In real-world scenarios, there are many network providers. We modify our two-party PSI-CA protocol to support heatmap computation between the HHS and multiple network providers without revealing additional information.

Second, we study associated rule learning (ARL) as an application of DotProd. ARL is a rule-based machine learning method that is used to discover rules/relations of the type $(X \Rightarrow Y)$ between variables X, Y in databases. As a typical example in the sales database of a supermarket, a rule/relation {onions, potatoes ⇒ burger} indicates that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat. In market design, such information can be used as the basis for decisions about product placements, promotional pricing, and more. However, the ARL training process requires a large transaction database, which may be collected from different sources. Thus, it is highly desirable to maintain the privacy of each source. We study a common ARL training algorithm, called Apriori [3, 43], and adapt it to the privacy-preserving setting. Most steps in Apriori can be computed locally except a step in which the parties want to compute a confidence score of how many transactions across a joint database that contains all attributes/items in both X and Y. This step can be implemented by computing a sum of bit-wise products of multiple binary vectors. We first apply multi-party DotProd for ARL and make its learning process in a privacy-preserving manner.

2 Preliminaries

Computational and statistical security parameters are denoted by κ, λ, respectively. We use $[x]$ to denote the set $\{1, 2, \ldots, x\}$ and $[x, y]$ to denote the set $\{x, x + 1, \ldots, y\}$. A set is a collection of distinct. We denote the concatenation of two bit strings x and y by $x||y$. For a pseudorandom function (PRF) F, a key k and a set A, we define $F(k, A) = \{F(k, a) \mid a \in A\}$. For an m-element binary vector x, we define $\mathbf{idx}(x) = \{i \in [m] \mid x[i] = 1\}$.

The functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{Coin}}$ for coin tossing between any number of parties is defined by $\{\perp, \perp, \ldots, \perp\} \mapsto x$ where x is drawn uniformly from $\mathcal{D}$. Depending on $\mathcal{D}$, the result from $\mathcal{F}^{\mathcal{D}}_{\mathsf{Coin}}$ can be used as a PRF key or a random value in any format. Secure protocol that computes $\mathcal{F}^{\mathcal{D}}_{\mathsf{Coin}}$ can be achieved in the dishonest majority setting by [29, 33].

2.1 Security Model

Secure computation allows mutually untrusted parties to jointly compute a function on their private inputs without revealing any additional information. There are two classical security models: colluding model is modeled by considering a single monolithic adversary that captures the possibility of collusion between the dishonest participants; and non-colluding model is modeled by considering independent adversaries, each captures the view of each independent dishonest party. There are also two adversarial models, which are usually considered. In the semi-honest (passive) model, the adversary is assumed to follow the protocol, but may try to learn information from the protocol transcript. In the malicious (active) model, the adversary follows an arbitrary polynomial-time strategy to learn additional information.

This paper introduces two variations of PSI-CA, each providing distinct security guarantees. Firstly, the server-aided variant of PSI-CA ensures security in the presence of an adversary who may passively corrupt any subset of $\{P_1, P_3, \ldots, P_n\}$ or $\{P_2, P_3, \ldots, P_n\}$ (i.e. P_1 and P_2 do not collude) or passively corrupt the server $\mathcal{C}$. The "server-less" protocol guarantees security in the presence of an adversary who may passively corrupt any subset from $\{P_3, \ldots, P_n\}$ or passively corrupt one of P_1,P_2 or P_n.

2.2 Oblivious Key-Value Store (OKVS)

A Key Value Store (KVS) consists of two algorithms: i) Encode takes as input a set of (k_i, v_i) key-value pairs from the key-value domain, $\mathcal{K} \times \mathcal{V}$, and outputs an object S (or, with negligible probability, an error indicator $\perp$); ii) Decode takes as input an object S, a key x and outputs a value y.

A KVS is correct if, for all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys: i) $Pr[\mathsf{Encode}(A) = \perp]$ is negligible, and ii) if $\mathsf{Encode}(A) = S \neq \perp$ and $(k, v) \in A$ then $\mathsf{Decode}(S, k) = v$.

EXPERIMENT 2.2.1. ($\mathsf{Exp}^{\mathcal{A}}(\mathcal{K} = (k_1, \ldots, k_m))$)

1. for $i \in [m]$: choose uniform $v_i \leftarrow \mathcal{V}$
2. return $\mathcal{A}\big(\mathsf{Encode}(\{(k_1, v_1), \ldots (k_m, v_m)\})\big)$

We say that a KVS is oblivious if for all $\mathcal{K}_1, \mathcal{K}_2$ of size m and all PPT adversaries $\mathcal{A}$: $\big|\Pr[\mathsf{Exp}^{\mathcal{A}}(\mathcal{K}_1) = 1] - \Pr[\mathsf{Exp}^{\mathcal{A}}(\mathcal{K}_2) = 1]\big| = \frac{1}{2} + \varepsilon$ where $\varepsilon \leq \mathsf{negl}(\kappa)$. In other words, if the values v_i are chosen uniformly then the output of Encode hides the choice of the keys k_i. Oblivious Key-Value Store (OKVS)[22] is given in Experiment 2.2.1, where $\mathcal{A}$ is an arbitrary PPT algorithm.

2.3 Oblivious PRF (OPRF) and Programmable PRF (OPPRF)

An oblivious PRF (OPRF) [20] is a 2-party protocol in which the sender learns a PRF key k and the receiver learns $F(k, q_1), \ldots, F(k, q_m)$. Here, F is a PRF and $(q_1, \ldots, q_m)$ are inputs chosen by the receiver. Functionality 1 presents a variant of OPRF where the receiver obtains outputs of multiple statically chosen queries.

FUNCTIONALITY 1. (*Oblivious PRF -* $\mathcal{F}^{m}_{\mathsf{oprf}}$)

Parameters: A PRF F, and a bound m on the number of queries.
Behavior: Wait for distinct queries $(q_1, \ldots, q_m)$ from the receiver where $q_i \in \{0,1\}^\kappa$. Sample a random PRF key k and give it to the sender. Give $\{F(k, q_1), \ldots, F(k, q_m)\}$ to the receiver.

An oblivious programmable PRF (OPPRF) [31] functionality is given in Functionality 2. It is similar to the plain OPRF functionality except that (1) it allows the sender to initially provide a set of points $\mathcal{P}$ which will be programmed into the PRF; (2) it additionally gives the public auxiliary information "hint" value to the receiver. Depending on the underlying OPPRF construction, the "hint" can be the random polynomial or bloom filter. For example, for polynomial construction, the program is done by interpolating the key-value pair into a polynomial and sending the coefficient as a hint so that the receiver can evaluate it. Other constructions may lead to hints of different forms. In general, a hint can be viewed as a data structure that allows the receiver to evaluate his input while not leaking any information for the sender's set. For further details about the hint, we direct the reader to [31, 37].

FUNCTIONALITY 2. (*Oblivious Programmable PRF -* $\mathcal{F}^{m_1,m_2}_{\mathsf{opprf}}$)

Parameters: A PRF F, an upper bound m_1 on the number of points to be programmed, and a bound m_2 on the number of queries.
Behavior: Wait for points $\mathcal{P} = \{(a_1, t_1), \ldots, (a_{m_1}, t_{m_1})\}$, with distinct keys a_i's, from the sender $\mathcal{S}$, and distinct queries $(q_1, \ldots, q_{m_2})$ from the receiver $\mathcal{R}$. Run $(k, \mathsf{hint}) \leftarrow \mathsf{KeyGen}(\kappa, \mathcal{P})$. Give (k, hint) to $\mathcal{S}$ and $(\mathsf{hint}, F(k, \mathsf{hint}, q_1), \ldots, F(k, \mathsf{hint}, q_{m_2}))$ to $\mathcal{R}$, where "hint" is the public auxiliary information.

2.4 Unconditional Zero Sharing

The unconditional zero sharing provides the parties with a sharing function $S : \{0,1\}^\kappa \times \{0,1\}^\ell \rightarrow \{0,1\}^\kappa$ and a key K_i for party P_i, such that for every $x \in \{0,1\}^\ell$, we have that $s_i = S(K_i, x)$ is P_i's random share, and $\bigoplus_{i=1}^{n} s_i = 0$. The functionality and its construction from [31] are given in Functionality 3 and Protocol 17.

FUNCTIONALITY 3. (*Zero-Sharing* - $\mathcal{F}_{\mathsf{ZS}}$)

Parameters: n parties. The dictionary store is initialized to $\emptyset$.
Behavior: P_i obtains a zero-sharing key K_i for a sharing function S. Upon an input x from P_i, if store_x does not exist, generate random values $s_1, \dots, s_n$ where $s_i = S(K_i, x)$ s.t. $\bigoplus_{i=1}^{n} s_i = 0$ and store $\mathsf{store}_{x,i} = s_i$ for $i \in [n]$. Output $K_i, \mathsf{store}_{x,i}$ to P_i.

2.5 Private Set Intersection Cardinality

Private set intersection cardinality (PSI-CA) allows n parties, each holding a set of m items, to learn the intersection size of their private sets without revealing anything else. In the server-aided PSI-CA, we assume there is a distrusted server that has no input and does not collude with the parties. The server is involved in the PSI-CA protocol while learning nothing. PSI-CA and server-aided PSI-CA are formally presented in Functionality 4. The highlighted text is required for the server-aided case.

FUNCTIONALITY 4. (*PSI Cardinality* - $\mathcal{F}_{\mathsf{PSI-CA}}$)

Parameters: n parties $P_1, \dots, P_n$; an untrusted server $\mathcal{C}$; the set size m.
Behavior:

- Wait for input set X_i of m distinct items from P_i.
- Give the server $\mathcal{C}$ nothing .
- Give P_1 an intersection set size $|\bigcap_{i=1}^{n} X_i|$.

2.6 Secure Dot Product of Binary Vectors

Secure dot product functionality allows n parties, each holding an m-element binary vector, to learn the dot product of their private vectors without revealing any additional information. In this work, we consider the problem of the secure dot product of n binary vectors, in a server-aided setting, in which we make use of a non-colluding distrusted server. Our protocols are extremely efficient when the upper bound on the Hamming weight of the vectors, denoted t, is in $o(m)$. The dot product of n vectors $x_1, \dots, x_n$, each with m elements, is defined by $\sum_{j=1}^{m} \prod_{i=1}^{n} x_i[j]$ and is called DotProd. DotProd is presented in Functionality 5. The highlighted text is required for the server-aided case.

FUNCTIONALITY 5. (*Secure Dot Product* - $\mathcal{F}_{\mathsf{DotProduct}}$)

Parameters: n parties: $P_{i \in [n]}$; an untrusted server $\mathcal{C}$; an upper-bound t.
Behavior:

- Wait for input m-element binary vector x_i from P_i.
- Give the server $\mathcal{C}$ nothing .
- Give to $\sum_{j=1}^{m} \prod_{i=1}^{n} x_i[j]$ the party P_1.

3 Server-Aided OPRF and OPPRF

In this section, we introduce new OPRF and OPPRF constructions which make use of a semi-honest non-colluding cloud server.

3.1 Server-Aided Shuffled OPRF

The server-aided OPRF functionality involves a sender $\mathcal{S}$, a receiver $\mathcal{R}$ and a server $\mathcal{C}$. It is defined as follows: $\mathcal{S}$ has a key-pair $k = (k_1, k_2)$ where $k_i \in \{0,1\}^{\kappa}$, $\mathcal{R}$ has a set of queries $\{y_i\}_{i \in [m]}$ and the server $\mathcal{C}$ has no input. $\mathcal{S}$ does not receive an output whereas $\mathcal{R}$ obtains one of the keys, specifically

k_1, and $\{y'_{\pi(1)}, \ldots, y'_{\pi(m)}\}$ where $y'_i = F'(k, y_i)$ and $\pi : [m] \to [m]$ is a random permutation. The output of $\mathcal{C}$ is one of the keys, specifically k_2, and the permutation π . Clearly, $\mathcal{R}$ cannot associate the response y'_i with the query y_i as all responses are pseudorandom. Figure 6 formally presents the ideal functionality of $\mathcal{F}^{(m)}_{\mathsf{soprf}}$.

FUNCTIONALITY 6. (*Server-Aided Shuffled OPRF* - $\mathcal{F}^{(m)}_{\mathsf{soprf}}$)

Parameters: $\mathcal{S}$, $\mathcal{R}$ and $\mathcal{C}$, the set size m, a pseudorandom permutation $F' : \{0,1\}^{2\kappa} \times \{0,1\}^{\ell} \to \{0,1\}^{\ell}$ where $F'(k_1, k_2, x) = F(k_2(F(k_1, x))$ where F is a PRP.

Behavior: Upon receiving a key $k = (k_1, k_2) \in (\{0,1\}^{\kappa})^2$ from $\mathcal{S}$.

- Give $\mathcal{C}$ a key k_2, and a random permutation $\pi : [m] \to [m]$.
- Give $\mathcal{R}$ the key k_1.

Then, upon receiving a set of *distinct* queries $\{y_i\}_{i \in [m]}$ from $\mathcal{R}$, send $\{F(k_1, y_i)\}_{i \in [m]}$ to $\mathcal{C}$ and send $\{y'_{\pi(1)}, \ldots, y'_{\pi(m)}\}$ to $\mathcal{R}$ where $y'_i = F'(k, y_i)$.

We first define $F'((k_1, k_2), x) = F(k_2, F(k_1, x))$ where F is a PRF. It is easy to see that F' is a PRF. In protocol $\Pi^{(m)}_{\mathsf{soprf}}$, the $\mathcal{S}$ has the key $k = (k_1, k_2)$, so it can send k_1 to $\mathcal{R}$ and k_2 to $\mathcal{C}$ as a part of their protocol's output. Having k_1, $\mathcal{R}$ computes $Y' = F(k_1, Y)$ and sends Y' to $\mathcal{C}$. The server $\mathcal{C}$ then computes $Y'' = F(k_2, Y')$, and applies a random permutation π on Y''. This protocol takes into account the presence of a semi-honest sender and a semi-honest receiver.

Theorem 1. *Protocol $\Pi^{(m)}_{\mathsf{soprf}}$ securely implements its functionality $\mathcal{F}^{(m)}_{\mathsf{soprf}}$ in the presence of an adversary who may passively corrupt either $\mathcal{S}$, $\mathcal{R}$, or $\mathcal{C}$.*

The formal proof of Theorem 1 is present in Appendix A.1.

PROTOCOL 7. (*Server-Aided Shuffled OPRF* - $\Pi^{(m)}_{\mathsf{soprf}}$)

Parameters:

- Set size m; a pseudorandom permutation (PRP) F.
- A sender $\mathcal{S}$, a receiver $\mathcal{R}$, a non-colluding semi-honest server $\mathcal{C}$

Inputs:

- Sender $\mathcal{S}$ has input $k = (k_1, k_2)$
- Receiver $\mathcal{R}$ has input a set of m items $Y = \{y_1, \ldots, y_m\}$
- Cloud server $\mathcal{C}$ has no input.

Protocol:

1. $\mathcal{S}$ sends k_1 to $\mathcal{R}$ and k_2 to $\mathcal{C}$.
2. $\mathcal{R}$ computes $Y' = F(k_1, Y)$ and sends Y' to $\mathcal{C}$.
3. $\mathcal{C}$ computes $Y'' = F(k_2, Y')$ and sends a random permutation π of Y'' to $\mathcal{R}$.

3.2 Server-Aided OPPRF

The server-aided OPRF functionality involves a sender $\mathcal{S}$, a receiver $\mathcal{R}$, and a non colluding server $\mathcal{C}$. It is defined as follows: $\mathcal{S}$ has a set of m_1 points $\mathcal{P} = \{(x_i, v_i)\}_{i \in [m_1]}$ with (pseudo)random v_i's, and $\mathcal{R}$ has a set $Y = \{y_i\}_{i \in [m_2]}$. $\mathcal{C}$ has no input. Denote the set of first (resp. second) entries of the pairs in $\mathcal{P}$ by X (resp. V). $\mathcal{S}$ and $\mathcal{C}$ do not have an output whereas $\mathcal{R}$, for every y_i, obtains v_i iff $y_i \in X$, and some other pseudorandom value otherwise. This is denoted by $\mathcal{F}_{\mathsf{sopprf}}$ and formally described in Functionality 8.

FUNCTIONALITY 8. (*Server-Aided OPPRF* - $\mathcal{F}^{(m_1,m_2)}_{\mathsf{sopprf}}$)

Parameters: $\mathcal{S}$, $\mathcal{R}$ and $\mathcal{C}$, m_1 the size of $\mathcal{P}$ and m_2 the number of queries.

Behavior:

- Wait for a set of m_1 points $\mathcal{P} = \{(x_i, v_i)\}_{i\in[m_1]}$ with distincts x_i's and (pseudo)random v_i's, from $\mathcal{S}$.
- Wait for a set $Y = \{y_i\}_{i\in[m_2]}$ from $\mathcal{R}$.
- For every $i \in [m_2]$ set $v'_i = v_j$ if $y_i = x_j$ for some $j \in [m_1]$ and otherwise assign a random value to v'_i. Let $V' = \{v'_i\}_{i\in[m_2]}$.
- Send V' to $\mathcal{R}$.

In the protocol, $\mathcal{S}$, $\mathcal{R}$ and $\mathcal{C}$ invoke a *non-shuffled* version of OPRF, where $\mathcal{S}$ inputs the key $k = (k_1, k_2)$, $\mathcal{R}$ inputs Y, and a sets $Y' = \{y'_1, \ldots, y'_{m_2}\}$ as a part of the output with $y'_i = F'(k, y_i)$. Then, $\mathcal{S}$ constructs an OKVS $T \leftarrow \mathsf{Encode}(\{(x_i, F'(k, x_i) \oplus v_i\}_{i\in[m_1]})$ and sends T to $\mathcal{R}$, which outputs $w_j = y'_j \oplus \mathsf{Decode}(T, y_j)$ for $j \in [m_2]$. In terms of the correctness, $\mathcal{R}$ obtains $v'_j = F'(k, y_j)) \oplus \mathsf{Decode}(T, y_j)$ for all $y_j \in Y$. If $y_j = x_i$, then $\mathsf{Decode}(T, y_i) = F'(k, x_i) \oplus v_j$, thus, $v'_j = v_i$. Otherwise, $\mathsf{Decode}(T, y_i)$ gives $\mathcal{R}$ a pseudorandom value which makes v'_j pseudorandom as well.

PROTOCOL 9. (*Server-Aided OPPRF* - $\Pi^{(m_1,m_2)}_{\mathsf{sopprf}}$)

Parameters:

- Parties are sender $\mathcal{S}$, receiver $\mathcal{R}$, and a server $\mathcal{C}$. Set sizes m_1, m_2. A PRP $F' : \{0,1\}^{2\kappa} \times \{0,1\}^{\ell} \to \{0,1\}^{\ell}$ where $F'(k_1, k_2, x) = F(k_2(F(k_1, x))$ where F is a PRF.

Inputs:

- $\mathcal{S}$ has $\mathcal{P} = \{(x_i, v_i)\}_{i\in[m_1]}$ with (pseudo)random v_i's.
- $\mathcal{R}$ has the set $Y = \{y_i\}_{i\in[m_2]}$.
- $\mathcal{C}$ has no input.

Protocol:

1. $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{C}$ jointly invoke $\mathcal{F}^{(m_2)}_{\mathsf{soprf}}$ where $\mathcal{S}$ inputs a random key $k = (k_1, k_2) \leftarrow \{0,1\}^{2\kappa}$, by which $\mathcal{C}$ obtains k_2 and $\mathcal{R}$ obtains k_1. Then $\mathcal{R}$ inputs Y, and obtains $Y' = \{y'_1, \ldots, y'_{m_2}\}$ as output, where $y'_i = F'(k, y_i)$. *Note that we use a non-shuffled version of OPRF.*
2. $\mathcal{S}$ constructs an OKVS over $T \leftarrow \mathsf{Encode}(\{(x_i, F'(k, x_i) \oplus v_i\}_{i\in[m_1]})$ and sends T to $\mathcal{R}$.
3. For every $j \in [m_2]$, $\mathcal{R}$ outputs $v'_j = y'_j \oplus \mathsf{Decode}(T, y_j)$.

Theorem 2. *Protocol $\Pi^{(m_1,m_2)}_{\mathsf{sopprf}}$ securely computes functionality $\mathcal{F}^{(m_1,m_2)}_{\mathsf{sopprf}}$ in the $\mathcal{F}^{(m)}_{\mathsf{soprf}}$-hybrid model, in the presence of an adversary who may passively corrupt either $\mathcal{S}$, $\mathcal{R}$, or $\mathcal{C}$.*

The formal proof of Theorem 2 is present in Appendix A.2.

4 PSI Cardinality Protocol

In this section we present three protocols:

- In Section 4.1, we simplify the server-aided PSI protocol of [28] and formally present a new server-aided two-party PSI-CA protocol. Unlike previous "server-less" protocols (see Section 1.1) that are based on oblivious transfer [18] or on the Diffie Hellman proble [46, 26], which in turn are based on public-key primitives, our two-party PSI-CA protocol uses only symmetric-key operations. This is possible, among other improvements, due to the replacement of their OPRF constructions with a server-aided version, which is much simpler and more efficient.
- In Section 4.2, we show an extension of the protocol to the multiparty case, where the adversary may passively corrupt (almost) any strict subset of the parties or passively corrupt the server. To

the best of our knowledge, this is the first 'special-purpose' protocol for privately computing the intersection cardinality of more than two parties, for which we present interesting applications (see Section 5).

- In Section 4.3, we show that a server is not necessary when some parties are assumed to be semi-honest and non-colluding.

4.1 Server-Aided Two-Party PSI-CA

We consider sender $\mathcal{S}$ and receiver $\mathcal{R}$ who want to compute the intersection size of their private sets $X = \{x_1, \ldots, x_{m_1}\}$ and $Y = \{y_1, \ldots, y_{m_2}\}$, respectively. To do so, they use a non-colluding, semi-honest cloud server $\mathcal{C}$. The formal description is given in Protocol 10. The protocol is inspired by the size-hiding server-aided PSI of Kamara et al. [28]. For completeness, a description of their PSI protocol is given in Appendix E.

For correctness, notice that for a value $z \in X \cap Y$, the value $F(k, z)$ appears in both X' and Y'. On the other hand, if $z \notin X$ then $F(k, z) \notin X'$; and if $z \notin Y$ then $F(k, z) \notin Y'$. The protocol is extremely efficient because of the efficiency of the shuffled $\mathcal{F}_{\mathsf{soprf}}$. In terms of communication cost, it only requires $\mathcal{S}$ to send m_1 values to $\mathcal{R}$. The construction for $\mathcal{F}_{\mathsf{soprf}}$, in turn, requires only m_2 messages from $\mathcal{R}$ to $\mathcal{C}$ and m_2 messages back from $\mathcal{C}$ to $\mathcal{R}$. We prove the following:

PROTOCOL 10. (*Server-Aided Two-party PSI-CA*)

Parameters:

- The protocol runs between a sender $\mathcal{S}$, a receiver $\mathcal{R}$, and a server $\mathcal{C}$. $\mathcal{S}$ and $\mathcal{R}$ have input size of m_1 and m_2, resp. A PRP $F' : \{0,1\}^{2\kappa} \times \{0,1\}^{\ell} \to \{0,1\}^{\ell}$.

Inputs:

- Sender $\mathcal{S}$ has input $X = \{x_1, \ldots, x_{m_1}\}$
- Receiver $\mathcal{R}$ has input $Y = \{y_1, \ldots, y_{m_2}\}$
- Cloud server $\mathcal{C}$ has no input.

Protocol:

1. $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{C}$ jointly invoke $\mathcal{F}_{\mathsf{soprf}}^{(m_2)}$ as follows: $\mathcal{S}$ inputs a random key $k = (k_1, k_2) \in \{0,1\}^{2\kappa}$, upon which $\mathcal{R}$ obtains k_1 and $\mathcal{C}$ obtains k_2 and π (recall that $\pi : [m_2] \to [m_2]$ is a random permutation). Then $\mathcal{R}$ inputs Y and obtains $Y' = \{y'_{\pi(1)}, \ldots, y'_{\pi(m_2)}\}$, where $y'_{\pi(i)} = F'(k, y_i)$.
2. $\mathcal{S}$ sends a random permutation of $X' = F'(k, X)$ to $\mathcal{R}$.
3. $\mathcal{R}$ outputs $|X' \cap Y'|$.

Theorem 3. *Protocol 10 securely implements Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) with $n = 2$ in the $\mathcal{F}_{\mathsf{soprf}}$-hybrid model, in the presence of an adversary who may passively corrupt either $\mathcal{S}$, $\mathcal{R}$, or $\mathcal{C}$.*

The formal proof of Theorem 3 is present in Appendix A.3.

4.2 Server-Aided Multi-Party PSI-CA

In this section, we assume that all parties have the same set size m. Protocol 11 may be seen as if we have one receiver, who is P_1, and multiple senders, who are $P_2, \ldots, P_n$. The role of the server is to shuffle PRF results from the senders before delivering them to the receiver. As a simplification to Protocol 11, suppose that we want the receiver to obtain $n - 1$ shares of zero for each of its items that is in the intersection. This can be done by querying the senders on each of their items and collecting the results. Each sender programs the responses such that if the query is on one of its items, then it responds with its (pseudorandom) share of zero, otherwise, it responds with some other pseudorandom value. Given the senders' responses on a query, if they sum up to zero then the receiver knows that its query is in the intersection. Since the server shuffles the responses to the queries, the receiver does not know, for a given set of responses which are shares of zero, to which query it is associated, thus, the output leaks nothing but the intersection size. Formally,

1. $P_2, \ldots, P_n$ (the senders) generate keys for a zero sharing function S, so P_i obtains K_i such that for every x it holds that $\bigoplus_{i=2}^{n} S(K_i, x) = 0$.
2. P_1 (the receiver) sends to the server its queries X_1.
3. The server runs an OPPRF instance with every sender, using the queries X_1. A sender P_i ($i \in [2, n]$) programs the responses such that on query $x \in X_i$ the response is $S(K_i, x)$ whereas on any other query the response is another pseudorandom value.
4. The server obtains the set $Y'_{i\in[2,n]}$, of $n-1$ OPPRF responses, on every query $x_i \in X_1$. It chooses a random permutation $\pi : [m] \to [m]$ and sends to P_1 the set $\{Y'_{\pi(1)}, \ldots, Y'_{\pi(m)}\}$.
5. P_1 checks for every response set Y_i whether its values are valid shares of zero. If so, it adds 1 to the cardinality.

In the above simplification, there are several security issues: first, the server learns P_1's queries in the clear; second, the server mediates all PRF responses and therefore it learns whenever there is a set of responses that are valid shares of zero, thus it can learn the intersection size as well; third, if the receiver colludes with one of the senders, together they can reverse the server's permutation on items that are in the intersection and by that leak the intersection itself (rather than only its size).

PROTOCOL 11. (*Server-Aided Multi-Party PSI-CA*)

Parameters:

- The protocol runs between parties $P_1, \ldots, P_n$ for $n > 2$, and a cloud server $\mathcal{C}$. A PRP $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$.

Inputs:

- P_i has $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$.
- Cloud server $\mathcal{C}$ has no input .

Protocol:

1. Parties $P_2, \ldots, P_n$ invoke $\mathcal{F}_{\mathsf{ZS}}$ (Functionality 3 with no input) and each party P_i obtains the key K_i for a sharing function S.
2. Parties P_1 and P_2 agree on m random values $\Gamma = (\gamma_1, \ldots, \gamma_m)$ using $\mathcal{F}_{\mathsf{Coin}}$.
3. Parties $P_1, \ldots, P_n$ agree on a random PRF key k using $\mathcal{F}_{\mathsf{Coin}}$.
4. Party P_i for $i \in [2, n]$ computes the set of points $\mathcal{P}_i$ where:
   - $\mathcal{P}_2 = \{(F(k, x_{2,j}), S(K_2, x_{2,j}) \oplus \gamma_{\pi(j)})\}_{j\in[m]}$ where $\pi : [m] \to [m]$ is a random permutation chosen by P_2.
   - For $i \in [3, n]$, $\mathcal{P}_i = \{(F(k, x_{i,j}), S(K_i, x_{i,j}))\}_{j\in[m]}$.
5. P_1 sends $X'_1 = F(k, X_1) = \{F(k, x_{1,j})\}_{j\in[m]}$ to $\mathcal{C}$.
6. $\mathcal{C}$ and P_i (for every $i \in [2, n]$) invoke $\mathcal{F}_{\mathsf{opprf}}$, where P_i acts as a sender with input $\mathcal{P}_i$ and $\mathcal{C}$ acts as a receiver with input X'_1. $\mathcal{C}$ obtains the result $y_{i,j}$ on the query $x_{1,j}$.
7. For every $j \in [m]$, $\mathcal{C}$ computes $w_j = \bigoplus_{i=2}^{n} y_{i,j}$ and sets W to be a random permutation of $\{w_1, \ldots, w_m\}$. $\mathcal{C}$ sends W to P_1.
8. P_1 outputs $|W \cap \Gamma|$.

The first issue is easily solved by having all parties $P_1, \ldots, P_n$ agree on a PRF key k, so instead of computing $|\bigcap_{i=1}^{n} X_i|$ their objective is to compute $|\bigcap_{i=1}^{n} F(k, X_i)|$. This way, the server does not know P_1's set. Hiding the intersection size from the server (the second issue above) is trickier. We solve it by having P_1 and P_2 agree on a set of random values $\Gamma = \{\gamma_1, \dots, \gamma_m\}$ so that instead of programming the responses with the 'zero shares', on a value $x \in X_2$, P_2 programs the response $S(K_2, x) \oplus \gamma$ for some $\gamma \in \Gamma$. Now, for items that are in the intersection, the server $\mathcal{C}$ sees a set of responses that constitutes a valid share of some $\gamma \in \Gamma$, but since the $\mathcal{C}$ does not know Γ, this looks random indistinguishable from the responses on values that are not in the intersection. Finally, we propose a protocol under a relaxed setting, that solves the third issue above. Concretely, the protocol is secure as long as P_1 and P_2 do not collude. This is done by adding one step to the above description: before the server forwards the responses set W to P_1, it sums its items and forwards only the sum to P_1. This means that now P_i ($i \geq 3$) could not trace back and learn the intersection itself. This is formally presented in Protocol 11.

Theorem 4. *Protocol 11 securely computes Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) for arbitrary n, in the ($\mathcal{F}_{\mathsf{opprf}}, \mathcal{F}_{\mathsf{ZS}}, \mathcal{F}_{\mathsf{Coin}}$)-hybrid model, in the presence of an adversary who may passively corrupt any subset of $\{P_1, P_3, \dots, P_n\}$ or $\{P_2, P_3, \dots, P_n\}$, or passively corrupt the server $\mathcal{C}$.*

We note that, in our protocol, parties use zero shares to mask their actual input. This step is similar to the one in [22]. The formal proof of the Theorem 4 is present in Appendix A.4.

PROTOCOL 12. (*Multi-Party PSI-CA*)

Parameters:

- The protocol runs between parties $P_1, \dots, P_n$ for $n > 2$, and a cloud server $\mathcal{C}$. A PRP $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$.

Inputs: P_i has $X_i = \{x_{i,1}, \dots, x_{i,m}\}$.

Protocol:

1. Parties $P_2, \dots, P_n$ invoke $\mathcal{F}_{\mathsf{ZS}}$ (Functionality 3) and each party P_i obtains the key K_i for a sharing function S.
2. Parties P_1 and P_2 agree on a random PRF key s using $\mathcal{F}_{\mathsf{Coin}}$.
3. Parties $P_2, \dots, P_n$ agree on a random PRF key k using $\mathcal{F}_{\mathsf{Coin}}$.
4. Party P_i for $i \in [2, n-1]$ computes the set of points $\mathcal{P}_i$ where:
   - $\mathcal{P}_2 = \left\{ \left(F(k, x_{2,j}), S(K_2, x_{2,j}) \oplus F(s, x_{2,j}) \right) \right\}_{j \in [m]}$.
   - For $i \in [3, n-1]$, $\mathcal{P}_i = \left\{ \left(F(k, x_{i,j}), S(K_i, x_{i,j}) \right) \right\}_{j \in [m]}$.
5. P_n and P_i (for every $i \in [2, n-1]$) invoke an instance of the server-aided OPPRF $\mathcal{F}_{\mathsf{sopprf}}^{(m,m)}$ where:
   - P_i acts as a sender with input $\mathcal{P}_i$,
   - P_1 acts as a cloud server with no input
   - P_n acts as a receiver with input $X'_n = F(k, X_n)$. P_n obtains the result $y_{i,j}$ on the query $x_{n,j}$.
6. For every $j \in [m]$, P_n computes $w_j = \bigoplus_{i=2}^{n-1} y_{i,j} \oplus S(K_n, x_{n,j})$. Then, P_n sets W to be $\{w_1, \dots, w_m\}$.
7. P_1 and P_n invoke the server-aided $\mathcal{F}_{\mathsf{PSI-CA}}$ functionality with P_2 as a server, where
   - P_n acts as a sender with input W
   - P_2 acts as a cloud server with no input
   - P_1 acts as a receiver with input $V = F(s, X_1)$, and obtains $|W \cap V|$.

4.3 Multi-party PSI-CA

We now describe our "server-less" multi-party PSI-CA protocol. The main idea is to convert the problem of n-party server-aided PSI-CA to the problem of $(n-1)$-party with the use of an untrusted party P_n who, however, has a private input set X_n. Recall that in the server-aided PSI-CA protocol, the cloud server $\mathcal{C}$ has no input, but obtains from P_1 the PRF values $F(k, X_1)$ which are used to invoke an OPPRF with parties $P_{i\in[2,n]}$. In the problem of $(n-1)$-parties, however, party P_n (who plays the role of $\mathcal{C}$) *does have* input X_n. Thus, P_n can compute its PRF values $F(k, X_n)$ on its own since it knows k. Similar to the server-aided version, P_n computes the exclusive-or of the OPPRF results and its zero share $S(K_n, x_{n,j})$, with the j-th result denoted by w_j. Note that w_j is equal to γ_j if all parties $P_{i\in[2,n]}$ has $x_{n,j}$, otherwise, w_j is random. At this point, if P_n sends all values w_j to P_1, P_1 can only compute the intersection size of $n-1$ sets $\bigcap_{i=2}^{n} X_i$ since there was nothing to do with the input set X_1.

Instead, to have P_1 output $|\bigcap_{i=1}^{n} X_i|$, we propose the following steps. Instead of using a random set Γ in Step (2) of Protocol 11, P_2 uses PRF to compute $\gamma_j \leftarrow F(s, x_{2,j})$ where s is known by only P_1 and P_2. We observe that if $x_{1,j}$ is an intersection item, the corresponding PRF value $F(s, x_{1,j})$ should be equal to a value w_k hold by P_n because of $w_k = \gamma_k = F(s, x_{2,k}) = F(s, x_{1,j})$. Therefore, the intersection size $|\bigcap_{i=1}^{n} X_i|$ can be computed by counting how many PRF values $F(s, x_{1,j})$ are in the set $W = \{w_1, \ldots, w_n\}$. P_1 and P_n can do this by invoking a two-party PSI-CA, where P_1 acts as a receiver with an input set $\{F(s, X_1)\}$ and P_n acts as a sender with an input set W.

We implement the two-party PSI-CA using our server-aid protocol described in Protocol 10 in which any party $P_{i\in[2,n-1]}$ (say P_2) can play the role of the cloud server. The two party PSI-CA Protocol 10 requires that both sender and receiver do not collude with the semi-honest server. Thus, in our multi-party protocol, we assume that P_2 is semi-honest and non-colluding with both P_1 and P_n. In addition, given this assumption, we can improve the performance of our multi-party OPPRF. Particularly, unlike Protocol 11 in the above section, we use our server-aided OPPRF construction described in Section 3.2 to execute an OPPRF instance between P_n and each $P_{i\in[2,n-1]}$, where P_1 plays the role of the OPPRF server (thus, P_1 is non-colluding). We formally present our server-less multi-party PSI-CA in Protocol 12, and its security statement below (see the formal proof in Appendix A.5).

Theorem 5. *Protocol 12 securely computes Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) for arbitrary n, in the ($\mathcal{F}_{\mathsf{sopprf}}, \mathcal{F}_{\mathsf{ZS}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{PSI-CA}}$, server-aided two-party PSI-CA)-hybrid model, in the presence of an adversary who may passively corrupt any subset from $\{P_3, \ldots, P_n\}$ or passively corrupt P_1 or P_2 (i.e. P_1 and P_2 are non-colluding).*

5 Applications

We demonstrate that our PSI-CA can be used for several privacy-preserving applications by implementing two running example applications which are built on our two-party and multi-party PSI-CA protocols, respectively.

5.1 Secure Dot Product Construction

Given a secure protocol for computing the cardinality of the intersection of the parties' sets, the protocol for dot product is simple. Let x_i be an m-element binary vector of party P_i, and let

$A_i = \mathbf{idx}(x_i)$. It is easy to see that the dot product of the x_i's is exactly the cardinality of the intersection of the A_i's, that is, $\sum_{j=1}^{m} \prod_{i=1}^{n} x_i[j] = |\bigcap_{i=1}^{n} A_i|$. Thus, to securely compute the dot product, we can use the PSI-CA functionality described in the previous section. Note that even though the input size is $O(m)$, the communication complexity of the protocol is only $O(t)$, which makes it extremely efficient when $t = o(m)$, where t is the upper bound on the Hamming weight of the vectors.

One subtle issue is that in the PSI-CA protocols the parties know the number of elements in each other's set, which leaks more information than required. Here, we assume that there is a known upper bound, t, on the Hamming weight of the vectors X_i's, and require that the parties' input to the PSI-CA contains exactly t items. That is, if the Hamming weight of X_i is $t' < t$ then P_i adds random "dummy" items to its input to the PSI-CA. Formally, for a given upper bound t, P_i inputs A_i to the PSI-CA where $A_i \leftarrow \mathbf{idx}'(X_i, t)$ and $\mathbf{idx}'(X, t)$ is defined as follows: let t' be the Hamming weight of X, set $A = \mathbf{idx}(X)$, pick $t - t'$ random values $D = \{d_1, \ldots, d_{t-t'}\}$ from the domain $\mathcal{D} = \{m+1, \ldots, 2^{\lambda+\log(t)} + m\}$ and output $A = A \cup D$. The choice of the domain $\mathcal{D}$ allows the collision probability of dummy items to be negligible and equals to $2^{-\lambda}$.

The formal description is given in Protocol 15 in Appendix D. Note that it is possible to compute dot product DotProd with or without the help of a cloud server $\mathcal{C}$, so both variants are presented. The protocol's correctness, complexity and security follow directly from the underlying PSI-CA protocol presented in Section 4 with different corruption structures.

Theorem 4. *Protocol 15 securely computes Functionality 5 ($\mathcal{F}_{\mathsf{DotProduct}}$) in the ($\mathcal{F}_{\mathsf{PSI-CA}}$)-hybrid model. In particular, if π is a protocol that securely computes $\mathcal{F}_{\mathsf{PSI-CA}}$ in the presence of an adversary $\mathcal{A}$ then, when instantiated with π, Protocol 15 is secure in the presence of adversary $\mathcal{A}$ as well.*

5.2 Heatmap Computation

As stated in [4], the heatmap can be considered as a two-party computation between HHS and a mobile network operator (MNO). HHS has a list of individuals who have reported positive for the disease. MNO knows an approximated location data of their subscribers as the subscriber connects to a certain cell tower when traveling (unless the user does not have a phone or disconnects to their network provider). Mathematically, HHS generates a binary vector $x \in \mathbb{Z}_2^N$ which indicates whether the user $i \in [1, N]$ amongst N subscribed individuals has tested positive ($x[i] = 1$) or not ($x[i] = 0$). For each cell tower $j \in [1, m]$, the MNO initializes a vector y_j of n elements, where $y_j[i]$ corresponds to the i-th subscriber (say that HHS and MNO agree on the subscribers' identifier and on their positions in the vectors). If the i-th subscriber connects to a cell tower j within some period of time, then $y_j[i] = 1$, and $y_j[i] = 0$ otherwise. To learn how many individuals visit a certain area (e.g. the area covered by the j-th cell tower, HHS and MNO run a secure dot product protocol to obtain $x \cdot y_j$.

The solution proposed in [4] relies on HE to implement the secure dot product for the heatmap problem. Even with the HE optimizations, [4] requires $O(N)$ independent secure multiplications to compute $x \cdot y_j$ for each cell tower. Therefore, their protocol costs $O(mN)$ HE multiplications to compute secure vector-matrix multiplications $x \cdot Y$, where Y consists of m columns $y_1, \ldots, y_m$. Each element of $x \cdot Y$ corresponds to how many diagnosed subscribers visited a cell town.

In this work, we observe that the proportion of diagnosed individuals among all N subscribed individuals is usually small (e.g. $0.01 - 0.1\%$ new positive cases per day [2]), thus, the vector x is sparse. In addition, the vector y_j is also sparse due to people's localized travel habits. Therefore, the heatmap computation is a perfect application for our $\mathsf{DotProd}$ where the input vectors are sparse. By applying $\mathsf{DotProd}$, we show that the computational complexity of the dot product in the heatmap example can be reduced from $O(N)$ to $O(t)$, where t is the maximum between the upper bound on the number of new positive test cases and the upper bound on the number of individuals visiting a geographical area covered by a cell tower.

PROTOCOL 13. (*Server-aided Heatmap Construction*)

PARAMETERS:
- Parameters k, N, t.
- A HHS and n MNO $P_1, \dots, P_n$, and a cloud server $\mathcal{C}$
- A PRF $F : \{0,1\}^\kappa \times \{0,1\}^\star \to \{0,1\}^\kappa$

INPUTS:
- A HHS P_0 has input a binary vector x of length N
- Each MNO $P_{k \in [n]}$ has input a binary matrix Y_k of size $N \times m$
- Cloud server $\mathcal{C}$ has no input.

PROTOCOL $n = 1$: For each $j \in [m]$, the HHS and the MNO P_1 invoke $\mathsf{DotProd}$ where P_0 input is x and $P1$ input is y^1_j. The HHS outputs $x \cdot y^1_j$

PROTOCOL $n > 1$:
1. HHS computes a set $A \leftarrow \mathbf{idx}(x)$ and pads A with dummy items to the upper-bound set size t.
2. $P_0, P_1, \dots, P_n$ agree on a random PRF key s using $\mathcal{F}_{\mathsf{Coin}}$.
3. For each $j \in [m]$:
   - (a) $P_{k \in [n]}$ computes a set $B_k \leftarrow \mathbf{idx}(y^k_j)$, and pads B_k with dummy items to the upper-bound set size t.
   - (b) Each MNO $P_{k \in [n]}$, the HHS, and the cloud server $\mathcal{C}$ jointly invoke a modified shuffled-OPRF:
     - P_k chooses two PRF keys $s_{k,1}, s_{k,2} \leftarrow \{0,1\}^\kappa$
     - P_k sends $s_{k,1}$ to HHS and sends $s_{k,1}$ to $\mathcal{C}$
     - HHS computes and sends $A'_k = F(s_{k,1}, A)$ to $\mathcal{C}$.
     - $\mathcal{C}$ computes $A''_k = F(s_{k,2}, A'_k)$.

     $\mathcal{C}$ sends a *permutation* of $A'' \leftarrow \{A''_1, \dots, A''_n\}$ to HHS
   - (c) Each $P_{k \in [n]}$ sends $B'''_k = F\big(s, F(s_{k,2}, F(s_{k,1}, B_k))\big)$ to $\mathcal{C}$ who sends a *permutation* of $B^\star \leftarrow \{B'''_1, \dots, B'''_n\}$ to HHS.
   - (d) HHS computes $A^\star = F(s, A'')$ and outputs $|A^\star \cap B^\star|$.

Multiple MNOs. We support a heatmap computation between one HHS, P_0, and multiple MNOs, $P_1, \dots, P_n$. For a cell tower $j \in [1, m]$, the MNO P_k ($k \in [n]$) has the vector y^k_j of N elements. $y^k_j[i]$ indicates whether a subscriber i connects to a cell tower j of the MNO P_k (we assume that the j-th cell tower of all MNOs covers the same geographical area, this should be adjusted in practice). The sum of the dot products $\sum_{k=1}^{n}(x \cdot y^k_j)$ indicates how many individuals, across different MNOs, visit a certain area. In our multi-party heatmap, if P_0 invokes $\mathsf{DotProd}$ with each MNO P_k where P_0's input is x and P_k's input is y^k_j, P_0 learns extra information – each term of the sum $\sum_{k=1}^{n}(x \cdot y^k_j)$. To address the issue, we modify the underlying shuffled-opprf protocol of $\mathsf{DotProd}$. At the high-level idea, $\mathcal{C}$ computes PRF values of all MNOs $P_{k \in [n]}$, permutes them before returning to the P_0. The formal description of our multi-party heatmap computation is presented in Protocol 13.

In real-world scenarios, HHS prefers to minimize bandwidth cost and computation workload on their side. Our protocol makes this happen by making use of the untrusted server. For the heatmap computation, HHS only needs to compute nmt and $2nmt$ symmetric-key operations in the two-party and multi-party settings, respectively. In terms of communication cost, HHS sends and receives $3nmt$ elements. Finally, our protocol requires only 1-round communication.

5.3 Association Rule Learning

Association rules learning (ARL) aims to discover regularities/rules between variables in transaction data. In this work, we use our DotProd protocol to mitigate information leakage in ARL when training the model on a vertical partitioning of the private database between multiple parties. We study the ARL definition in [3] and adapt it to the privacy-preserving context (see Definition 1 in Appendix B). We consider only a vertically-partitioned database since if the data is horizontally-partitioned, each party can locally compute ARL. For whom are not familiar with ARL, we provide a detailed explanation of the algorithm in Appendix C.

PROTOCOL 14. (*Privacy-Preserving* ARL)

PARAMETERS:

- A ARL threshold τ, α attributes, empty lists $L_n, \dots, L_\alpha$.
- n parties: $P_1, \dots, P_n$.
- An DotProd functionality described in Functionality 5.
- An `apriori-gen` algorithm described in Figure 1.

INPUTS: $P_{i\in[n]}$ has input a vertically-partitioned database $T_{i\in[n]}$.

PROTOCOL:

1. $P_{i\in[n]}$ locally computes a list L_1^i of frequent itemsets that has only 1 attribute.
2. $P_{i\in[n]}$ invoke a DotProd with each attribute input $j_i \in L_1^i$, and add j_i into a published list L_n if the output of the DotProd is great than τ (e.g. a sum of element-wise products of multiple sparse binary vectors $T[j_i]$ as $\sum_{v=1}^{m} \prod_{i=1}^{n} T[j_i][v] > \tau$)
3. For $k = n+1$ to α, if L_k is empty, the parties do the following:
   (a) $P_{i\in[n]}$ locally computes $C_k = \texttt{apriori-gen}(L_{k-1})$.
   (b) For each candidate $c \in C_k$, let $J = \{j_1, \dots, j_m\}$ be a set of attributes in c.
      - Assume that each $P_{i\in[n]}$ have h_i attributes $J_i = \{j_{i_1}, \dots, j_{i_{h_i}}\}$. P_i locally computes an element-wise product of multiple binary vectors $T[j_{i_v}]$ as $X_i \leftarrow \prod_{v=1}^{h_i} T[j_{i_v}]$
      - Parties invoke a DotProd execution:
      – P_i inputs X_i.
      – P_1 obtains the output s, and adds c to L_{k+1} if $s > \tau$

Privacy-preserving ARL (PPARL) consists of two subproblems (see Appendix B). The second subproblem can be publicly solved since the frequent itemsets are a part of the ARL result. According to [47], one can reduce the first subproblem of PPARL to securely computing the dot products of the binary vectors with minor leakage information. For simplicity, consider the candidate itemset has only two attributes. Let x and y represent columns in the database. i.e., $x[i] = 1$ iff row i has value 1 for attribute X (similar for y and Y). Each party P_1 and P_2 holds a vertically-partitioned database of the transaction x and y respectively. The dot product of two m-element vectors x and y as $x \cdot y = \sum_{i=0}^{m} x[i]y[i]$ is the support count which indicates how many times the itemset XY appears in the joint transaction set. The dot product computation requires the joint database from both parties, thus, it should be computed in a privacy-preserving manner. Given $s \leftarrow x \cdot y$, the parties can check whether the obtained support count is greater or equal to the threshold τ. If yes, the candidate itemset is a frequent itemset. In the ideal world, if $s < \tau$, the exact value of s is not revealed to the parties. Thus, the information is considered as leakage information in our PPARL scheme as well as previous work [47, 16]. Note that [47, 16] reveal more information than ours - they leak indexes that $x[i] = y[i] = 1$ (i.e. intersection items).

In this work, we consider n-party setting with *global* rules where every vertically-partitioned transaction database $T_{i\in[n]}$ has at least one item in the frequent itemset. Protocol 14 presents our PPARL construction which closely follows the Apriori algorithm [3, 47]. The first two steps aim to find a list of itemsets that (1) appear in the transaction set T at least τ times; and (2) every party has at least one attribute in the itemset. We denote the obtained list to be L_n. Given L_n, the party locally computes a list of candidates C_{n+1} for itemsets of size $n+1$ using the `apriori-gen` algorithm [3]. At the high-level idea, the function `apriori-gen` is done by generating a superset of possible candidate itemsets and pruning this set. We present the `apriori-gen` algorithm in Figure 1, and refer the reader to [3] for more detail. Note that `apriori-gen` is computed on the public list L_n, thus it leaks no additional information. The parties jointly execute Step (3) to compute $L_{t>n}$ until it is empty.

6 Implementation and Performance

We evaluate the performance of our PSI-CA (or DotProd) protocols and estimate the performance of heatmap computation and ARL. Protocols are evaluated under different network settings, number of parties, and input set sizes to demonstrate their scalability.

Choice of Parameters. We run experiments on a single machine 2× 36-core Intel Xeon 2.30GHz CPU and 256GB of RAM and simulated network using the Linux *tc* command. We consider two network settings: the LAN setting has 0.02ms round-trip latency and 10 Gbps network bandwidth; the WAN setting has 96ms round-trip latency and 200 Mbps network bandwidth. In our implementation, each party uses a separate thread to communicate with other parties. The computational security parameter $\kappa = 128$ and the statistical security parameter $\sigma = 40$. The number of parties is in a range of $\{2, 4, 8, 16\}$. The set size m of PSI-CA or the upper-bound Hamming weight t of DotProd is in $\{2^{12}, 2^{16}, 2^{20}, 2^{24}\}$.

Choice of PRF, OPPRF, and OKVS We instantiate the PRF F using AES-NI. We use OKVS and OPPRF as a black box in the implementation. Our implementation uses the table-based OPPRF code from [31]. While there are different OKVS constructions [22], we choose the most efficient Encode and Decode of 3-cuckoo PaXoS data structure. The number of bins in the cuckoo table is $1.3m$ with 3 hash functions.

PSI-CA and DotProd protocols. Recall that the steps of PSI-CA and DotProd protocols are similar, except for a small cost overhead in Step (1) of DotProd where each party locally computes a function **idx**(). In the DotProd protocol, we assume that there is a known upper bound, t, on the Hamming weight of the party's input vector X. To implement DotProd using PSI-CA, we require that the parties' input to the PSI-CA contains exactly t items. Thus, we only report the detailed computational and communication performance results of our PSI-CA protocols for the set size m. It indicates that the DotProd protocols are evaluated with the upper bound $t = m$.

Table 1: Run time (in second) and communication cost (in MB) of our "server-less" multiparty PSI-CA protocols for n parties on sets of size m.

	m	$n=4$				$n=8$				$n=16$			
		P_1	P_2	$P_{(3:n-1)}$	P_n	P_1	P_2	$P_{(3:n-1)}$	P_n	P_1	P_2	$P_{(3:n-1)}$	P_n
Runtime LAN (second)	2^{12}	0.07	0.07	0.06	0.07	0.07	0.07	0.06	0.07	0.08	0.08	0.06	0.08
	2^{16}	0.40	0.37	0.21	0.33	0.40	0.38	0.22	0.34	0.430	0.39	0.23	0.36
	2^{20}	6.01	5.69	3.99	6.38	6.32	5.77	4.26	7.02	6.75	6.43	4.60	7.16
Runtime WAN (second)	2^{12}	1.52	1.33	0.06	0.75	1.73	1.54	0.06	0.96	1.74	1.55	0.06	0.97
	2^{16}	4.21	3.61	0.98	2.37	4.59	4.00	1.17	2.76	6.09	5.49	1.46	4.26
	2^{20}	21.60	21.24	11.32	20.20	33.75	33.34	19.86	32.34	59.63	59.27	36.72	58.26
Comm. Cost (MB)	2^{12}	0.52	0.28	0.16	0.71	1.02	0.28	0.16	1.85	2.02	0.29	0.16	4.12
	2^{16}	8.27	4.54	2.54	11.35	16.27	4.54	2.54	29.51	32.27	4.54	2.54	65.83
	2^{20}	132.32	72.64	40.64	181.60	260.32	72.64	40.64	472.16	516.32	72.64	40.64	1053.28

Table 2: Run time (in second) and communication cost (in MB) of[11] and our protocols for 4 parties and no collusion. Each party has a set size m. The numbers of [11] are for PSI itself (not, PSI-CA).

	PSI [11] (server-less, semi-honest)			PSI-CA Protocol 11 (server-aided, semi-honest)			PSI-CA Protocol 12 (server-less, semi-honest)		
m	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
LAN	0.23	1.6	23.8	0.19	1.38	19.65	0.07	0.4	6.38
WAN	1.9	7	108.2	1.89	6.9	106.08	1.52	4.21	21.6
Comm.	3.2	49.4	790.2	3.41	53.86	967.32	0.84	13.35	213.6

Table 3: Run time (in second) and communication cost (in MB) of[36] and our server-less protocol for n parties. Each party has a set size m.

	Three-party PSI-CA [36]			Our PSI-CA Protocol 12					
(m,n)	$(8,2^{14})/(4,2^{15})$	$(8,2^{18})/(4,2^{19})$	$(8,2^{22})/(4,2^{23})$	$(8,2^{14})$	$(8,2^{18})$	$(8,2^{22})$	$(4,2^{15})$	$(4,2^{19})$	$(4,2^{23})$
LAN	0.2	3.1	74	0.14	1.72	28.16	0.14	1.68	27.20
WAN	1.8	15.8	267	2.37	13.19	134.29	2.21	9.25	104.01
Comm.	32.6	521.5	8344	7.88	187.00	1008.32	6.68	106.80	1708.80

6.1 Performance of Two-party Protocols

PSI-CA Protocol. We evaluate our two-party PSI-CA protocol in the LAN and WAN settings. We consider both balanced and unbalanced set sizes as our heatmap computation is built on the asymmetric two-party PSI-CA. In our protocol, the parties do not need to involve in the entire protocol's computation. The sender $\mathcal{S}$ send $F((k_1,k_2),X)$ and k_1 to the receiver, send k_2 to the server $\mathcal{C}$ at the same time and complete its computation. Similarly, the $\mathcal{C}$ does not need to be online during the whole process. Instead, the $\mathcal{C}$ start its computation when receiving the $\mathcal{S}$'s key PRF k_2 and the set of receiver's queries. Therefore, we report the performance of each participant separately in Table 6 (in Appendix). We find that our protocol scales well in the experiments as it contains only AES calls. For instance, the total run time of our PSI-CA with the input set size $m_1 = m_2 = 2^{20}$ is only 1.5 seconds.

Comparison with Prior Work. Both DH-based and delegated PSI-CA [18] protocols are secure against a semi-honest adversary, but the latter requires two non-colluding servers. Note that one can use the protocol proposed in [35] to implement PSI-CA, however, the protocol is much expensive compared to DH-based PSI-CA. The PSI-CA implementation of [46, 15] is not available[2], thus we omit to compare theirs with ours. In addition, we compare our protocol with ROOM-based protocol [44]. The two-party DotProd of [44] consists of two expensive steps: ROOM and a generic dense matrix multiplication. In Table 4, we only report the performance of ROOM in settings where [44] performs best.

We use DH-based PSI code implemented by [42] with the fastest Curve25519 implementation from `libsodium`. For a fair comparison, we run the implementation of delegated PSI-CA [18] and DH-based PSI on the same benchmark machine and network settings. Note that [18] only provides the implementation of their protocol building blocks, thus, there are no performance results on the WAN setting. The times[3] for ROOM are taken from [44, Figure 17] and [32, Table 2], initially provided for a database $50,000$ and a number of queries $5,000$ and $50,000$. Table 4 presents the performance of each PSI-CA protocol. When comparing the protocols, we find that the running time of our protocol is $10 - 100\times$ faster than that of the prior works. In addition, our protocol requires $2 - 5\times$ less bandwidth cost compared to them. The results show the benefit of using our protocols in a reasonable server-aided model.

Performance of Heatmap Computation. In the two-party setting, executing the heatmap computation essentially involves multiple DotProd or PSI-CA executions. Similar to [4], we want to evaluate our protocol for smaller nation-states such as New York City or Singapore which has a population around $N = 2^{23}$. Concretely, we consider a case in which the MNO has a matrix Y of size $N \times m$ and the HHS has a vector x of N, where $N = 2^{23}$ and $m = 2^{15}$. The parties need to perform m DotProd instances as $x \cdot y_{j\in[m]}$, where y_j is the j^{th} column of Y. Recall the x and y_j are binary vectors that indicate whether an individual tested positive to COVID-19, and whether this individual visited a place nearby the network town y_j, respectively. Among $N = 2^{23}$, we assume that there are $t_2 = 2^{12}$ new positive cases per day [2], and each patient visits 4 places per day on average. We run $m = 2^{15}$ instances of our two-party PSI-CA protocol with the MNO's set size $t_1 = 2^{14}$ and the HHS's set size $t_2 = 2^{12}$, and find that our protocol costs about 10 minutes using a *single* thread. On the other hand, [4] reports about 90 minutes but using 96 threads and stronger benchmark machine [4]. Therefore, we estimate that our protocol is at least $50\times$ faster than [4]. It dues to the fact that our protocol is based on symmetric-key operations while [4] heavily relies on public-key operations. In addition, [4] requires that the participants agree on database indices (i.e. data alignment before running heatmap computation). Using PSI-CA, we can remove this requirement. The party's input can be a set of patient/visitor ids (instead of the vector/matrix).

Performance of ARL Based on the DotProd performance, we *estimate* the performance of our ARL. In two-party setting, each party $P_{i\in[2]}$ locally computes a list L_1^i of frequent itemsets that has only one attribute. The parties sequentially invoke DotProd to compute lists L_k of frequent itemsets that has exactly k attributes where L_{k+1} is empty (say L_{m+1} is empty). Assume that each attribute/vector in $L_{j\in[2,m]}$ has a Hamming weight t_j. Also, assume that each C_j has $|C_j|$ candidates. The performance of our ARL is $\sum_{i=2}^{m}|C_j|[\Pi^{(t_j,2)}_{\mathsf{DotProduct}}]$, where $[\Pi^{(t_j,2)}_{\mathsf{DotProduct}}]$ is the cost

[2] [46] requires a non-colluding server that is similar to ours, but their protocol heavily replies on DH based PSI. [15] requires two non-colluding senders, each holds an identical input set.

[3] Unknown benchmark machine

[4] an c5.24xlarge AWS EC2 instance (96 vCPU @ 3.6 GHz, 192 GiB RAM)

of two-party DotProd with Hamming weight t_j. According to Table 6, we estimate that our ARL would take under hours to compute ARL of the database with million records.

6.2 Performance of Multi-party Protocols

PSI-CA Protocol. The running times and communication overhead of our server-aided multi-party PSI-CA are shown in Table 5 (Appendix). The protocol is asymmetric with respect to the server, the receiver P_1 and other parties $P_{i\in[2,n]}$, thus, we report the performance results of these parties separately. In our protocol, the workload of the receiver is light as it only requires to call m AES instances. The majority of the receiver's running time is to wait for other parties to finish their work. For example, P_1 takes 33.86 seconds to compute PSI-CA (or DotProd) with $n = 8$ and $m = 2^{20}$ (or $t = 2^{20}$) in the LAN setting. Also, the server plays the role of the receiver in most OPPRFs, his communication cost is highest amongst other participants. For $n = 8$ and $m = 2^{20}$ (or $t = 2^{20}$), the protocol PSI-CA (or DotProd) requires 3305 MB on the server's side.

Table 1 presents the performance of our "server-less" multiparty PSI-CA protocol in both LAN and WAN settings. Similar to the server-aided protocol, we separately report the performance results of P_1, P_2, P_n and other parties $P_{i\in[3,n-1]}$. Unlike server-aided protocol, this protocol only relies on OKVS (i.e. makes use of symmetric-key operations only). We find that our protocol scales to large input sets (e.g. $m = 2^{20}$) with a large number of participants (e.g. $n = 16$). For $n = 16$ and $m = 2^{20}$ (or $t = 2^{20}$), our protocol requires only 6 seconds with the total communication cost 1GB.

Comparison with Prior Work. The three-party PSI-CA protocol [36] can be applied to multi-party cases by letting all the n parties secret-share their set of m items to their three parties/leaders S_1, S_2, S_3, then the three leaders jointly compute the PSI-CA output. The three leaders conduct the computation in the honest-majority model, which might achieve the similar security assumption in our server-less protocol in which P_1, P_2, P_n acts as leaders. To implement a n-party PSI-CA, each having m input items, the protocol of [36] requires to run PSI-CA on the total of mn secret-shared input items. Note that [36] only consider computing the PSI-CA for two sets, each of m items. Thus, the running time and communication cost of their protocol reported in [36, Figure 8] is for computing PSI-CA on the total of $2m$ secret-shared input items. To have a fair comparison, we report the performance of ours and [36]'s protocol for the total mn input items. For example, computing PSI-CA for $n = 2^3$ parties, each with $m = \{2^{14}, 2^{18}, 2^{22}\}$, results in the computation of the total $mn \in \{2^{17}, 2^{21}, 2^{25}\}$ elements. This is equivalent to the experiential results for the two-party PSI-CA using [36] with the set size $\{2 * 2^{16}, 2 * 2^{20}, 2 * 2^{24}\}$, which are reported in [36, Figure 8] where each party has $\{2^{16}, 2^{20}, 2^{24}\}$ input items, respectively (i.e., one needs to execute the two-party PSI-CA of [36] with each input set of $mn/2$ items). Since the implementation of [36] is not publicly available, we take numbers from the publication and have the comparison with our protocol. We present the detailed performance comparison in Table 3[5]. Our protocol shows about 2.5× faster than [36] for sufficient large m. We also note that when these leaders servers collude, our protocol only reveals the intersection items while [36] leaks all input items to the adversary.

As far as we know, [11]'s implementation is not publicly available. Thus, we take their reported run times from [11, Table 2-5]. For the most direct comparison, we used the same configured machine (2x 36-core Intel Xeon 2.30GHz 256GB of RAM) and network settings to evaluate their and our protocols. We compare our "server-less" protocol with [11] for the case of $n = 4$, one

[5] we estimate the running time by linear interpolation

dishonestly colluding (no collusion), each with $m \in \{2^{12}, 2^{16}, 2^{20}\}$. We show an improvement of $1.6 - 5\times$ in the run time, and $3.5 - 4\times$ in the bandwidth cost. We report the performance numbers in Table 2. Our server-less protocol with $n = 16$ requires only 6.38s in the LAN setting and $m = 2^{20}$ (see Table 1). From Table 2, the [11] with $n = 4$ requires 23.8s in the same setting. Our protocol with $n = 16$ is already 3.74× faster than [11] with $n = 4$, thus, we do not present the comparison of the two protocols for larger n.

Performance of Heatmap Computation. The complexity of our heatmap protocol is linear in the number of MNOs. Using the suitable parameters of the two-party heatmap where each MNO has a matrix of size $2^{23} \times 2^{15}$, and HHS has a vector of size 2^{23}, we *estimate* that our protocol takes about one hour if there are 6 MNOs involved in the protocol execution. Note that our protocol does not reveal additional information other than the output – how many patients visit a certain area. In contrast, [4] only works in the two-party setting. In real-world scenarios, there are many MNOs. If using only their protocol where the HHS executes vector-matrix multiplication with each MNO and then computes the "global" heatmap, this solution leaks extra information – the individual result of each vector-matrix multiplication.

Performance of ARL Similar to the two-party ARL, the performance of our multi-party ARL is $\sum_{i=n}^{m} |C_j| [\Pi^{(t_j,n)}_{\mathsf{DotProduct}}]$, where $[\Pi^{(t_j,n)}_{\mathsf{DotProduct}}]$ is the cost of n-party DotProd with Hamming weight t_j. Here, we assume that each attribute/vector in $L_{j\in[n,m]}$ has a Hamming weight t_j. According to the performance of our multi-party DotProd (or multi-party PSI-CA) shown in Table 5&1, we estimate that our ARL would take under a day to compute ARL of the database with million records.

References

[1] Outsourcing scalar products and matrix products on privacy-protected unencrypted data stored in untrusted clouds. *Information Sciences*, 2018.

[2] Covid-19 coronavirus pandemic, 2021. =https://www.worldometers.info/coronavirus/.

[3] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.

[4] A. Bampoulidis, A. Bruni, L. Helminger, D. Kales, C. Rechberger, and R. Walch. Privately connecting mobility to infectious diseases via applied cryptography. Cryptology ePrint Archive, Report 2020/522, 2020. https://ia.cr/2020/522.

[5] D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *LNCS*, pages 420–432. Springer, 1991.

[6] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. STOC, 1990.

[7] A. Berke, M. Bakker, P. Vepakomma, R. Raskar, K. Larson, and A. Pentland. Assessing disease exposure risk with location histories and protecting privacy: A cryptographic approach in response to a global pandemic. *arXiv preprint arXiv:2003.14412*, 2020.

[8] A. Bhowmick, D. Boneh, S. Myers, K. Talwar, and K. Tarbe. The apple psi system, 2021. [Online; accessed 18-Sept-2021].

[9] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Heidelberg, Apr. 2015.

[10] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, Oct. 2016.

[11] N. Chandran, N. Dasgupta, D. Gupta, S. L. B. Obbattu, S. Sekar, and A. Shah. Efficient linear multiparty psi and extensions to circuit/quorum psi. CCS, 2021.

[12] M. Chase and P. Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 34–63. Springer, Heidelberg, Aug. 2020.

[13] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Heidelberg, Aug. 2007.

[14] D. Demmler, T. Schneider, and M. Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS 2015*. The Internet Society, Feb. 2015.

[15] S. Dittmer, Y. Ishai, S. Lu, R. Ostrovsky, M. Elsabagh, N. Kiourtis, B. Schulte, and A. Stavrou. Function secret sharing for psi-ca: With applications to private contact tracing. Cryptology ePrint Archive, Report 2020/1599, 2020.

[16] C. Dong and L. Chen. A fast secure dot product protocol with application to privacy preserving association rule mining. In *PAKDD*, 2014.

[17] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: an efficient and scalable protocol. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 2013*, pages 789–800. ACM Press, Nov. 2013.

[18] T. Duong, D. H. Phan, and N. Trieu. Catalic: Delegated PSI cardinality with applications to contact tracing. In S. Moriai and H. Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 870–899. Springer, Heidelberg, Dec. 2020.

[19] E. Fenske, A. Mani, A. Johnson, and M. Sherr. Accountable private set cardinality for distributed measurement. *ACM Trans. Priv. Secur.*, 25(4), jul 2022.

[20] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In J. Kilian, editor, *TCC*, 2005.

[21] G. Garimella, P. Mohassel, M. Rosulek, S. Sadeghian, and J. Singh. Private set operations from oblivious switching. In J. A. Garay, editor, *Public-Key Cryptography – PKC 2021*, pages 591–617, Cham, 2021. Springer International Publishing.

[22] G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. Oblivious key-value stores and amplification for private set intersection. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 395–425, Virtual Event, Aug. 2021. Springer, Heidelberg.

[23] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[24] S. Goldwasser and S. Micali. Probabilistic encryption how to play mental poker keeping secret all partial information. STOC '82, 1982.

[25] C. Hu, R. Li, W. Li, J. Yu, Z. Tian, and R. Bie. Efficient privacy-preserving schemes for dot-product computation in mobile computing. PAMCO '16, 2016.

[26] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *EuroS&P*, pages 370–389. IEEE, 2020.

[27] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert. Mobile private contact discovery at scale. In *USENIX*, August 14-16, 2019.

[28] S. Kamara, P. Mohassel, M. Raykova, and S. S. Sadeghian. Scaling private set intersection to billion-element sets. In N. Christin and R. Safavi-Naini, editors, *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 195–215. Springer, 2014.

[29] J. Katz, R. Ostrovsky, and A. Smith. Round efficiency of multi-party computation with a dishonest majority. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 578–595. Springer, Heidelberg, May 2003.

[30] L. Kissner and D. X. Song. Privacy-preserving set operations. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, Heidelberg, Aug. 2005.

[31] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu. Practical multi-party private set intersection from symmetric-key techniques. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 1257–1272. ACM Press, Oct. / Nov. 2017.

[32] T. Lepoint, S. Patel, M. Raykova, K. Seth, and N. Trieu. Private join and compute from pir with default. Cryptology ePrint Archive, Report 2020/1011, 2020. https://ia.cr/2020/1011.

[33] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 171–189. Springer, Heidelberg, Aug. 2001.

[34] C. Meadows. Formal verification of cryptographic protocols: A survey (invited lecture). In J. Pieprzyk and R. Safavi-Naini, editors, *ASIACRYPT'94*, volume 917 of *LNCS*, pages 135–150. Springer, Heidelberg, Nov. / Dec. 1995.

[35] P. Miao, S. Patel, M. Raykova, K. Seth, and M. Yung. Two-sided malicious security for private intersection-sum with cardinality. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 3–33. Springer, Heidelberg, Aug. 2020.

[36] P. Mohassel, P. Rindal, and M. Rosulek. Fast database joins and PSI for secret shared data. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 1271–1287. ACM Press, Nov. 2020.

[37] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. PSI from PaXoS: Fast, malicious private set intersection. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Heidelberg, May 2020.

[38] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. Efficient circuit-based PSI with linear communication. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Heidelberg, May 2019.

[39] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In K. Fu and J. Jung, editors, *USENIX Security 2014*, pages 797–812. USENIX Association, Aug. 2014.

[40] P. Rindal and P. Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930. Springer, Heidelberg, Oct. 2021.

[41] M. Rosulek and L. Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, Aug. 2021. Springer, Heidelberg.

[42] M. Rosulek and N. Trieu. Compact and malicious private set intersection for small sets. CCS, 2021. https://ia.cr/2021/1159.

[43] C. Rudin. Mit lecture notes: Machine learning and statistics, 2012. https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec01.pdf.

[44] P. Schoppmann, A. Gascón, M. Raykova, and B. Pinkas. Make some ROOM for the zeros: Data sparsity in secure distributed machine learning. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 1335–1350. ACM Press, Nov. 2019.

[45] B. Siabi, M. Berenjkoub, and W. Susilo. Optimally efficient secure scalar product with applications in cloud computing. *IEEE Access*, 2019.

[46] N. Trieu, K. Shehata, P. Saxena, R. Shokri, and D. Song. Epione: Lightweight contact tracing with strong privacy. *arXiv*, 2020.

[47] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. KDD, 2002.

[48] J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13:593–622, 10 2005.

[49] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, Oct. 1986.

[50] J. Zhang, X. Wang, S.-M. Yiu, Z. Jiang, and J. Li. Secure dot product of outsourced encrypted vectors and its application to svm. 2017.

Table 4: Run time (in second), communication cost (in MB), and system requirement of the two-party PSI-CA (or DotProd) protocols: DH-based PSICA [26, 34], OSN-based PSI-CA [21], Catalic [18], ROOM as a building block in DotProd [44], and ours (a simpler variant of the [28] PSI protocol) for the sender set size m_1 and receiver set size m_2. Cells with $-$ denote trials that are not supported by the protocol.

	DH-PSICA [26]				OSN-based PSI-CA [21]				ROOM [44]				Catalic [18]				Ours			
m_1	2^{16}		2^{20}		2^{16}		2^{20}		2^{16}		2^{20}		2^{16}		2^{20}		2^{16}		2^{20}	
m_2	2^{12}	2^{16}	2^{16}	2^{20}	2^{12}	2^{16}	2^{16}	2^{20}	2^{12}	2^{16}	2^{16}	2^{20}	2^{12}	2^{16}	2^{16}	2^{20}	2^{12}	2^{16}	2^{16}	2^{20}
LAN	8.31	10.21	112.51	191.87	-	6.56	-	84.88	14.3	144.17	-	-	6.41	8.92	85.1	166.12	0.1	0.13	1.01	1.5
WAN	11.26	11.5	150.14	248.32	-	24.57	-	284.62	-	-	-	-	-	-	-	-	1.54	2.37	4.85	8.24
Comm.	2.82	4.78	46.14	77.59	-	55.49	-	1030	863	13788	878	13837	6.29	6.29	100.66	100.66	1.18	3.15	18.87	50.33
System Req.	server-less semi-honest parties												two non-colluding servers semi-honest parties/servers				one non-colluding server semi-honest parties/servers			

[51] Y. Zhu, Z. Wang, B. Hassan, Y. Zhang, J. Wang, and C. Qian. Fast secure scalar product protocol with (almost) optimal efficiency. In S. Guo, X. Liao, F. Liu, and Y. Zhu, editors, *Collaborative Computing: Networking, Applications, and Worksharing*, pages 234–242, Cham, 2016. Springer International Publishing.

A Correctness and Security Proof

A.1 Server-Aided Shuffled OPRF

Theorem 1. *Protocol $\Pi^{(m)}_{\mathsf{soprf}}$ securely implements its functionality $\mathcal{F}^{(m)}_{\mathsf{soprf}}$ in the presence of an adversary who may passively corrupt either $\mathcal{S}$, $\mathcal{R}$, or $\mathcal{C}$.*

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{S}}$, $\mathsf{Sim}_{\mathcal{R}}$, and $\mathsf{Sim}_{\mathcal{C}}$ for simulating the view of corrupt $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{C}$ respectively which consists of the randomness, input, output, and received messages during the execution of the protocol. And then we argue the indistinguishability of the produced transcript from the real execution.

- *Corrupted $\mathcal{S}$.* $\mathcal{S}$ does not receive anything during the execution of the protocol. So it is trivial to simulate his view.
- *Corrupted $\mathcal{R}$.* $\mathsf{Sim}_{\mathcal{R}}$ randomly select a pair of keys (k_1, k_2) and appends the k_1 to the view. Given the PRF F, $\mathsf{Sim}_{\mathcal{R}}$ computes $Y'' = F(k_2, F(k_1, Y))$ for the input set Y and appends a permutation of Y'' to the view. Now we argue that the view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one. The way that $\mathsf{Sim}_{\mathcal{R}}$ selects keys is identical to the real execution. Outputs of the PRF given different keys are computationally indistinguishable. So this simulated view is computationally indistinguishable from the real execution.
- *Corrupted $\mathcal{C}$.* $\mathsf{Sim}_{\mathcal{C}}$ randomly select a pair of keys (k_1, k_2) and appends the k_2 to the view. Given the PRF F, $\mathsf{Sim}_{\mathcal{C}}$ computes $Y' = F(k_1, Y)$ for the randomly selected input set Y and appends it to the view. Now we argue that the view output by $\mathsf{Sim}_{\mathcal{C}}$ is indistinguishable from the real one. The way that $\mathsf{Sim}_{\mathcal{C}}$ selects keys is identical to the real execution. Outputs of the PRF given are computationally indistinguishable. So this simulated view is computationally indistinguishable from the real execution.

□

A.2 Server-Aided OPPRF

Theorem 2. *Protocol $\Pi^{(m_1,m_2)}_{\mathsf{sopprf}}$ securely computes functionality $\mathcal{F}^{(m_1,m_2)}_{\mathsf{sopprf}}$ in the $\mathcal{F}^{(m)}_{\mathsf{soprf}}$-hybrid model, in the presence of an adversary who may passively corrupt either $\mathcal{S}$, $\mathcal{R}$, or $\mathcal{C}$.*

Table 5: Run time (in second) and communication cost (in MB) of our server-aided multiparty PSI-CA protocols for n parties on sets of size m.

	m	$n = 4$			$n = 8$			$n = 16$		
		P_1	$P_{(2:n)}$	Server	P_1	$P_{(2:n)}$	Server	P_1	$P_{(2:n)}$	Server
Runtime LAN (second)	2^{12}	0.19	0.18	0.19	0.35	0.28	0.35	0.54	0.39	0.54
	2^{16}	1.38	1.02	1.37	2.31	1.22	2.3	4.56	1.97	4.55
	2^{20}	19.65	15.6	19.39	33.86	16.8	33.61	71.17	32.36	70.89
Runtime WAN (second)	2^{12}	1.89	1.15	1.84	2.47	1.16	2.08	3.04	1.25	2.66
	2^{16}	6.9	3.18	6.01	14.07	4.1	13.28	26.09	6.01	25.3
	2^{20}	106.08	23.98	97.49	197.35	39.43	196.87	409.13	71.26	408.65
Comm. Cost (MB)	2^{12}	0.13	1.64	5.05	0.13	1.64	11.61	0.13	1.64	24.73
	2^{16}	2	25.93	79.79	2	25.93	183.51	2	25.93	390.95
	2^{20}	32	467.66	1434.98	32	467.66	3305.62	32	467.66	7046.9

Table 6: Running time (in second) and communication cost (in MB) of our two-party PSI-CA protocols for the sender set size m_1 and receiver set size m_2.

		Comm.			LAN			WAN		
m_2	m_1	Receiver	Sender	Server	Receiver	Sender	Server	Receiver	Sender	Server
2^8	2^8	0.012	0.004	0.008	0.002	0.001	0.002	0.481	0.001	0.289
	2^{10}	0.025	0.016	0.008	0.002	0.002	0.002	0.482	0.002	0.29
	2^{12}	0.074	0.066	0.008	0.008	0.006	0.006	0.679	0.005	0.486
2^{12}	2^{12}	0.197	0.066	0.131	0.01	0.005	0.008	1.066	0.005	0.681
	2^{14}	0.393	0.262	0.131	0.029	0.019	0.02	1.274	0.02	0.888
	2^{16}	1.18	1.049	0.131	0.099	0.065	0.065	1.537	0.066	1.144
2^{16}	2^{16}	3.146	1.049	2.097	0.132	0.058	0.102	2.374	0.065	1.579
	2^{18}	6.291	4.194	2.097	0.315	0.209	0.203	2.924	1.42	2.097
	2^{20}	18.874	16.777	2.097	1.007	0.583	0.553	4.853	3.732	3.597
2^{20}	2^{20}	50.332	16.777	33.554	1.501	0.964	1.206	8.235	5.745	7.681
	2^{22}	100.663	67.109	33.554	4.814	2.637	4.247	19.535	15.373	18.772
	2^{24}	301.99	268.435	33.554	19.123	9.625	17.305	66.244	54.594	64.089

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{S}}$, $\mathsf{Sim}_{\mathcal{R}}$, and $\mathsf{Sim}_{\mathcal{C}}$ for simulating the view of corrupt $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{C}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

- *Corrupted* $\mathcal{S}$. $\mathsf{Sim}_{\mathcal{S}}$ simulates the view of corrupt $\mathcal{S}$, which consists of $\mathcal{S}$'s randomness, input, output, and received messages. $\mathsf{Sim}_{\mathcal{S}}$ proceeds as follows. It chooses a random key $k = (k_0, k_1) \leftarrow \{0,1\}^{2\kappa}$, calls a $\mathsf{Sim}_{\mathcal{S}}^{\mathsf{soprf}}$ of the server-aided OPPRF, and appends its output to the view. Since the $\mathsf{Sim}_{\mathcal{S}}^{\mathsf{soprf}}$ is trivial, it is easy to see the view of $\mathsf{Sim}_{\mathcal{S}}$ is computationally indistinguishable from the real execution.
- *Corrupted* $\mathcal{R}$. $\mathsf{Sim}_{\mathcal{R}}$ simulates the view of corrupt $\mathcal{R}$, which consists of $\mathcal{R}$'s randomness, input, output, and received messages. $\mathsf{Sim}_{\mathcal{R}}$ proceeds as follows. It calls a $\mathsf{Sim}_{\mathcal{R}}^{\mathsf{soprf}}$ with input Y and appends the output to the view. To simulate Step 3, $\mathsf{Sim}_{\mathcal{R}}$ generates m_1 random points $(x_i, v_i) \leftarrow \{0,1\}^{\ell} \times \{0,1\}^{\ell}$, constructs an OKVS over $T \leftarrow \mathsf{Encode}(\{(x_i, v_i)\})$, and appends it to the view.

  We now argue that the output of $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real execution. For this, we formally show the simulation by proceeding with the sequence of hybrid transcripts $T0; T1; T2$, where $T0$ is the real view of S, and $T3$ is the output of $\mathsf{Sim}_{\mathcal{R}}$.

  - Hybrid 1. Let T_1 be the same as T_0, except the output of server-aid OPRF execution is replaced by the output of the $\mathsf{Sim}_{\mathcal{R}}^{\mathsf{soprf}}$. It is easy to see T_0 and T_1 are computationally indistinguishable.
  - Hybrid 2. Let T_2 be the same as T_1, except the OKVS T is constructed on randomly selected points (x_i, v_i). Since the value $F'(k, x_i) \oplus v_i$ are also pseudorandom in the real execution, the two constructed OKVS tables T are computationally indistinguishable.

- *Corrupted $\mathcal{C}$.* Since the $\mathcal{C}$ only participates in the execution of server-aid OPRF as the $\mathcal{C}$, the construction of $\mathsf{Sim}_{\mathcal{C}}$ can inherit from the $\mathsf{Sim}_{\mathcal{C}}$ in the proof of Theorem 1 directly. So it is computationally indistinguishable from the real execution and we omit the proof here.

□

A.3 Server-Aided Two-party PSI-CA

Theorem 3. *Protocol 10 securely implements Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) with $n = 2$ in the $\mathcal{F}_{\mathsf{soprf}}$-hybrid model, in the presence of an adversary who may passively corrupt either $\mathcal{S}$, $\mathcal{R}$, or $\mathcal{C}$.*

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{S}}$, $\mathsf{Sim}_{\mathcal{R}}$, and $\mathsf{Sim}_{\mathcal{C}}$ for simulating the view of corrupt $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{C}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

- *Corrupted $\mathcal{S}$.* $\mathsf{Sim}_{\mathcal{S}}$ simulates the view of corrupt $\mathcal{S}$, which consists of $\mathcal{S}$'s randomness, input, output, and received messages. $\mathsf{Sim}_{\mathcal{S}}$ proceeds as follows. It chooses a random key $k = (k_0, k_1) \leftarrow \{0,1\}^{2\kappa}$, calls a $\mathsf{Sim}_{\mathcal{S}}^{\mathsf{soprf}}$, and appends its output to the view. Since the $\mathsf{Sim}_{\mathcal{S}}^{\mathsf{soprf}}$ does not receive any messages in the protocol, it is easy to see the view of $\mathsf{Sim}_{\mathcal{S}}$ and the view in the real execution are identical.

- *Corrupted $\mathcal{R}$.* $\mathsf{Sim}_{\mathcal{R}}$ simulates the view of corrupt $\mathcal{R}$, which consists of $\mathcal{R}$'s randomness, input, output, and received messages. $\mathsf{Sim}_{\mathcal{R}}$ proceeds as follows. It calls a $\mathsf{Sim}_{\mathcal{R}}^{\mathsf{soprf}}$ with input Y and appends the output to the view. To simulate Step 2, $\mathsf{Sim}_{\mathcal{R}}$ generates a random set of m_1 values $X = \{x_1, ..., x_{m_1}\}$, chooses random key $k' = (k'_1, k'_2) \leftarrow \{0,1\}^{2\kappa}$, computes $X'' = F'(k, X)$ and appends it to the view. The PRF values X'' received from the simulator are computationally indistinguishable from the random permutation of X' received from the real execution.

- *Corrupted $\mathcal{C}$.* Since the $\mathcal{C}$ only participates in the execution of server-aid OPRF as the $\mathcal{C}$, the construction of $\mathsf{Sim}_{\mathcal{C}}$ can inherit from the $\mathsf{Sim}_{\mathcal{C}}$ in the proof of Theorem 1 directly. So it is computationally indistinguishable from the real execution and we omit the proof here.

□

A.4 Server-Aided Multi-Party PSI-CA

Correctness. We consider three following cases based on whether x is in the intersection of all sets $X_{i \in [n]}$:

- Case 1: Suppose $x \in \bigcap X_{i\in[n]}$. In other words, $\forall i \in [n], \exists x_{i,j_i} \in X_i$, such that $x_{i,j_i} = x$. Thus, we have (i) all PRF values $x'_{i,j_i} = F(k, x_{i,j_i}) = F(k, x)$ are equal, (ii) XORing all zero shares $S(K_i, x_{j_i}), i \in [2, n]$ is equal to zero. When querying the OPPRF programmed $\mathcal{P}_{i\in[3,n]}$ using the common PRF value $x'_1 = F(k, x)$, the cloud server obtains $y_{i,j}$. Based on the correctness of OPPRF, we have $y_{i,j_i} = S(K_i, x_{j_i})$. In addition, the cloud server obtains $y_{2,j_2} = S(K_2, x_{j_2}) \oplus \gamma_{j_2}$ when querying on x'_1. Therefore, the value $w_j = \bigoplus_{i=2}^{n} y_{i,j_i}$ is equal to γ_{j_2} which belongs to the set Γ known by P_1. Thus, P_1 can count how many w_j in Γ to output the intersection size.
- Case 2: Suppose x is in X_1 and is not an element in some sets $X_{i\in[2,n]}$. Some OPPRF output $y_{i,j}$ is a random value since $F(k, x)$ was never used in the OPPRF programming process. Therefore, $w_j = \bigoplus_{i=2}^{n} y_{i,j}$ is random and does not belong to the set Γ.
- Case 3: Suppose x is an element in some sets $X_{i\in[2,n]}$, but not in X_1. Some OPPRF output $y_{i,j}$ is a random value. Therefore, $w_j = \bigoplus_{i=2}^{n} y_{i,j}$ is random and does not belong to the set Γ.

Theorem 4. *Protocol 11 securely computes Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) for arbitrary n, in the ($\mathcal{F}_{\mathsf{opprf}}, \mathcal{F}_{\mathsf{ZS}}, \mathcal{F}_{\mathsf{Coin}}$)-hybrid model, in the presence of an adversary who may passively corrupt any subset of $\{P_1, P_3, \ldots, P_n\}$ or $\{P_2, P_3, \ldots, P_n\}$ or passively corrupt the cloud server $\mathcal{C}$.*

Proof. We separate the proof to the maximal collusion, from which a security to non-maximal ones can be derived. We exhibit simulators in three different cases and argue the indistinguishability of the produced transcript from the real execution.

- *Case 1: $P_1, P_3, \ldots, P_n$ are corrupted.* The simulator first calls the $\mathcal{F}_{\mathsf{ZS}}$ simulator $\mathsf{Sim}^{\mathsf{ZS}}$ and appends the parties keys K_i's for a zero sharing to the view of $P_1, P_3, \ldots, P_n$. In addition, the simulator calls the $\mathcal{F}_{\mathsf{Coin}}$ simulator $\mathsf{Sim}^{\mathsf{FCoin}}$, appends the set Γ to the view of P_1, and appends a PRF key k to the view of $P_1, P_3, \ldots, P_n$. The simulator now calls the $\mathcal{F}_{\mathsf{opprf}}$ simulator $\mathsf{Sim}^{\mathsf{OPPRF}}_{\mathcal{S}}$ with input $\mathcal{P}_i = \{(F(k, X_i), S(K_i, X_i))\}$ and appends the output to the view of $P_3, \ldots, P_n$. Then the simulator calls the $\mathcal{F}_{\mathsf{opprf}}$ simulator $\mathsf{Sim}^{\mathsf{OPPRF}}_{\mathcal{R}}$ with input $F(k, X_1) = \{F(k, x_{1,j})\}_{j\in[m]}$ for each instance of $P_3, \ldots, P_n$, receives $Y_i = \{y_{i,j}\}_{i\in[2,\ldots,n],j\in[m]}$, computes set $W = \{w_j\}_{j\in[m]}$ where $w_j = \bigoplus_{i=2}^{n} y_{i,j}$, and appends a random permutation of W to the view of P_1 ($\{y_{2,j}\}_{j\in[m]}$ are obtained by randomly choose m values from $\{0,1\}^{\ell}$). The joint view of the parties $P_1, P_3, \ldots, P_n$ is identically distributed in the simulation, and in the real execution, the messages seen by them are identically distributed and so is the output given to P_1 (who is the only party receiving output).

- *Case 2: $P_2, P_3, \ldots, P_n$ are corrupted.* Most of the simulation is similar to the case above. The simulator first calls the $\mathcal{F}_{\mathsf{ZS}}$ simulator $\mathsf{Sim}^{\mathsf{ZS}}$ and appends the parties keys K_i's for a zero sharing to the view of $P_2, \ldots, P_n$. In addition, the simulator calls the $\mathcal{F}_{\mathsf{Coin}}$ simulator $\mathsf{Sim}^{\mathsf{FCoin}}$, appends the set Γ to the view of P_2, and appends a PRF key k to the view of $P_2, \ldots, P_n$. The simulator now calls the $\mathcal{F}_{\mathsf{opprf}}$ simulator $\mathsf{Sim}^{\mathsf{OPPRF}}_{\mathcal{S}}$ with input $\mathcal{P}_i = \{(F(k, X_i), S(K_i, X_i))\}$ and appends the output to the view of $P_2, \ldots, P_n$. This concludes the simulation. The joint view of the parties is identically distributed in the simulation and in the real execution, the messages seen by them are identically distributed and these corrupted parties do not receive outputs.

- *Case 3: $\mathcal{C}$ is corrupted.* The server $\mathcal{C}$ has no input or output. In the protocol it receives the pseudorandom values X' from P_1 and the pseudorandom values $y_{i,j}$ for $i \in [2, n]$ and $j \in [m]$. These $m \cdot n$ values can be easily simulated by handing $\mathcal{C}$ $m \cdot n$ random values. The output of all parties $P_1, \ldots, P_n$ are identically distributed in the simulation and the real execution. It only remains to argue that the view of $\mathcal{C}$ is computationally indistinguishable in both cases, which follows from the security of the PRP F and the OPPRF functionality.

□

A.5 Multi-party PSI-CA

Correctness. We consider three following cases based on whether x is in the intersection of all sets $X_{i\in[n]}$:

- Case 1: Suppose $x \in \bigcap X_{i\in[n]}$. In other words, $\forall i \in [n], \exists x_{i,j_i} \in X_i$, such that $x_{i,j_i} = x$. Thus, we have (i) all PRF values $F(k, x_{i,j_i}) = F(k, x)$ are equal, (ii) XORing all zero shares $S(K_i, x_{j_i}), i \in [2, n]$ is equal to zero. When querying the OPPRF points $\mathcal{P}_{i\in[2,n]}$ using the common PRF value $F(k, x)$, the party P_n obtains y_{i,j_i}. Based on the correctness of OPPRF, we have $y_{i,j_i} = S(K_i, x)$ for $i \in [3, n-1]$ and $y_{2,j_2} = S(K_i, x) \oplus PRF(s, x)$. Therefore, the value $w = \big(\bigoplus_{i=2}^{n-1} y_{i,j_i}\big) \oplus S(K_n, x)$ is equal to $PRF(s, x)$ as $\bigoplus_{i=2}^{n} S(K_i, x) = 0$. Step (7) allows P_1 to count x to output the intersection set by checking whether $w \in F(s, X)$.
- Case 2: Suppose x is in X_1 and is not an element in some sets $X_{i\in[2,n]}$. Clearly, $w \notin F(s, X_1)$ with the high probability.
- Case 3: Suppose x is an element in some sets $X_{i\in[2,n]}$, but not in X_1. The value w_j might equal to $F(s, x_2)$ for $x_2 \in X$ or random. However, $x_2 \notin X_1$, thus $w \notin F(s, X_1)$ with the high probability.

Theorem 5. *Protocol 12 securely computes Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) for arbitrary n, in the ($\mathcal{F}_{\mathsf{sopprf}}, \mathcal{F}_{\mathsf{ZS}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{PSI-CA}}$, server-aided two-party PSI-CA)-hybrid model, in the presence of an adversary who may passively corrupt any subset from $\{P_3, \ldots, P_n\}$ or passively corrupt P_1 or P_2 (i.e. P_1 and P_2 are non-colluding).*

Proof. We separate the proof into multiple cases, depending on the adversary's corruption. As before, we assume maximal corruption and stress that the security in the case of non-maximal corruption can be easily derived. We exhibit simulators in three different cases and argue the indistinguishability of the produced transcript from the real execution.

- *Case 1: $P_3, \ldots, P_n$ are corrupted.* The simulator first calls the $\mathcal{F}_{\mathsf{ZS}}$ simulator $\mathsf{Sim}^{\mathsf{ZS}}$ and appends the parties keys K_i's for a zero sharing to the view of $P_3, \ldots, P_n$. In addition, the simulator calls the $\mathcal{F}_{\mathsf{Coin}}$ simulator $\mathsf{Sim}^{\mathsf{FCoin}}$, appends a PRF key k to the view of $P_3, \ldots, P_n$. The simulator now calls the $\mathcal{F}_{\mathsf{opprf}}$ simulator $\mathsf{Sim}^{\mathsf{OPPRF}}_{\mathcal{S}}$ with input $\mathcal{P}_i = \{(F(k, X_i), S(K_i, X_i))\}$ and appends the output to the view of $P_3, \ldots, P_{n-1}$. Then the simulator calls the $\mathcal{F}_{\mathsf{opprf}}$ simulator $\mathsf{Sim}^{\mathsf{OPPRF}}_{\mathcal{R}}$ with input $F(k, X_n) = \{F(k, x_{n,j})\}_{j\in[m]}$ for each instance of $P_2, \ldots, P_{n-1}$, receives $Y_i = \{y_{i,j}\}_{i\in[2,\ldots,n-1],j\in[m]}$, computes set $W = \{w_j\}_{j\in[m]}$ where $w_j = \bigoplus_{i=2}^{n-1} y_{i,j} \oplus S(K_n, x_{n,j})$. Then the simulator calls the two-party server-aided $\mathcal{F}_{\mathsf{PSI-CA}}$ simulator $\mathsf{Sim}^{\mathsf{PSI-CA}}_{\mathcal{S}}$ with input W, and appends the output to the view of P_n.

  This concludes the simulation. The joint view of $P_3, \ldots, P_n$ is computationally indistinguishable for the simulation and in the real execution.

- *Case 2: Corrupted P_2.* The simulator first calls the $\mathcal{F}_{\mathsf{ZS}}$ simulator $\mathsf{Sim}^{\mathsf{ZS}}$ and appends the key K_2's for a zero sharing to the view of P_2. In addition, the simulator calls the $\mathcal{F}_{\mathsf{Coin}}$ simulator $\mathsf{Sim}^{\mathsf{FCoin}}$, appends PRF keys s and k to the view of P_2. In the $\mathcal{F}^{(m,m)}_{\mathsf{sopprf}}$ invocation between P_2 and P_n the simulator obtains $\mathcal{P}_2$, from which it can extract X_2 (since it knows k and can invert F). The simulator now calls the $\mathcal{F}_{\mathsf{opprf}}$ simulator $\mathsf{Sim}_{\mathcal{S}}^{\mathsf{OPPRF}}$ with input $\mathcal{P}_2 = \{(F(k, X_2), S(K_2, X_2) \oplus F(s, X_2))\}$ and appends the output to the view of P_2. Finally to simulating the server-aided $\mathcal{F}_{\mathsf{PSI-CA}}$, the simulator calls the simulator $\mathsf{Sim}_{\mathcal{C}}^{\mathsf{SPSI-CA}}$ and appends the output to the view of P_2 This concludes the simulation. The view of P_2 is computationally indistinguishable for the simulation and in the real execution.

- *Case 3: Corrupted P_1.* The simulator first calls the $\mathcal{F}_{\mathsf{Coin}}$ simulator $\mathsf{Sim}^{\mathsf{FCoin}}$, appends a PRF key s to the view of P_2. Then the simulator calls the two-party server-aided OPPRF simulator $\mathsf{Sim}_{\mathcal{C}}^{\mathsf{SOPPRF}}$ without input, and appends the output to the view of P_1. Finally the simulator calls the simulator $\mathsf{Sim}_{\mathcal{R}}^{\mathsf{SPIS-CA}}$ with input $V = F(s, X_1)$ and appends the output to the view of P_1. This concludes the simulation. The view of P_1 is computationally indistinguishable for the simulation and in the real execution.

□

B Multi-party ARL

Definition 1. *In the privacy-preserving ARL (PPARL) problem, there are n parties $P_1, \ldots, P_n$, each holding a private vertically-partitioned database of transactions $T_1, \ldots, T_n$, respectively. Let $T = T_1 || \ldots || T_n$ be a jointed vertically database of n parties. Let $I = \{i_1, i_2, ..., i_m\}$ be a public set of binary attributes, called items. Each transaction (row) $t \in T$ is represented as a binary vector, with $t[k] = 1$ if the transaction contains item $i_k \in I$, and $t[k] = 0$ otherwise. We say that the transaction t satisfies* $\mathbf{idx}(t)$*. Denote an association rule by $\Rightarrow$. Let $X, Y \subseteq [m]$, we consider the following association rules:*

1. *The rule $X \Rightarrow Y$ holds in T with support factor of $0 \le s \le 1$ iff at least $s\%$ of transactions in T satisfy $X \cup Y$*
2. *The rule $X \Rightarrow Y$ holds in T with confidence factor of $0 \le c \le 1$ iff at least $c\%$ of transactions in T that satisfy X also satisfy Y.*
3. *The rule $X \Rightarrow Y$ is global if every transaction in T has at least one item in $X \cup Y$.*

The goal of PPARL is to allow all parties $P_1, \ldots, P_n$ to find all ***global*** *rules having high support and confidence on their jointed database T while maintaining the privacy of each individual database.*

Generally speaking, the support factor indicates how frequently the itemset appears in the dataset. The support of X with respect to T is defined as the proportion of transactions in the dataset which contains the itemset X. That is, $\mathrm{supp}(X) = \frac{|\{X \subseteq T\}|}{|T|}$.

The confidence factor indicates how often the rule $X \Rightarrow Y$ is true. The confidence value of a rule, $X \Rightarrow Y$, in a set of transactions T, is the proportion of the transactions that contain X which also contain Y. $\mathrm{conf}(X \Rightarrow Y) = \frac{\mathrm{supp}(X \cup Y)}{\mathrm{supp}(X)}$. Thus confidence can be interpreted as an estimate of the conditional probability.

Given the definitions of support and confidence factors, the method for finding an association rule [3] can be decomposed into two subproblems.

(1) Find the frequent itemset: The frequent itemset is defined as the itemset that appears in the transaction set T at least τ times, where τ is predefined minimum support (also called a threshold).

(2) Use the frequent itemsets to generate the association rules: For every large itemset X, find all non-empty subsets A of X. For every such subset A, output a rule of the form $A \Rightarrow (X \setminus A)$ if the ratio of supp(X) to supp(A) is at least τ.

C Example of the ARL algorithm

For simplicity, we consider two parties P_1 and P_2, each holding a vertical-partitioned database T_1 and T_2, respectively. Assume that T_1 has 3 attributes/columns $\{a_1, a_2, a_3\}$, and T_2 has 2 attributes/columns $\{b_1, b_2\}$.

One important step of the ARL algorithm is to find all "global" frequent itemsets. For example, we want to compute how many transactions that contain 2 attributes (a_1, b_1). If the number of these transactions is greater than a threshold t, we say that (a_1, b_1) is a frequent itemset.

For a better protocol explanation. We define "global" vs "local" frequent itemset. A frequent itemset is global if each party has at least one item in the frequent itemset (this aligns with the global rule mentioned in Definition 1). A frequent itemset is local if the frequent itemset contains only items belonging to one party.

If (a_1, b_1) is a "global" frequent itemset, the attribute a_1 itself should be a "local" frequent itemset. Thus, before any interaction between parties, each party P_i needs to locally compute a list L_1^i that has only 1 attribute. For example, if the attribute a_1 appears more than or equal t times in T_1, then a_1 is a local frequent itemset, and thus a_1 is added to L_1^1. In contrast, if the attribute a_2 appears less than t times in the T_1, then a_2 is not a local frequent itemset, and thus $a_2 \notin L_1^1$. Assume that from Step 1, we have $L_1^1 = \{a_1, a_3\}$, and $L_1^2 = \{b_1, b_2\}$.

Step 2 of Protocol 14 aims to find a list L_n of "global" frequent itemsets, where each itemset has n items ($n = 2$ in the two-party setting). To do so, the parties run $\mathsf{DotProd}$ where the party's input is each itemset in L_1^1 and L_1^2. For example, the parties check whether each of pairs $(a_1, b_1), (a_1, b_2), (a_3, b_1), (a_3, b_2)$ are "global" frequent items. Assume that the column a_1 is $(1, 1, 1, 0, 0)$ and the column b1 is $(1, 1, 1, 1, 0)$. The dot product $a_1 \cdot b_1$ is 3. E.g. a pair (a_1, b_1) appears 3 times in the database. If the threshold $t = 2$, the (a_1, b_1) is a "global" frequent itemset.

Step 3 of the protocol aims to find a list L_k of "global" frequent itemsets, where each itemset has k items (here, $k > 2$). For example, the parties want to check whether (a_1, a_3, b_1) is a "global" frequent itemset (in this case, $k = 3$). They first need to compute the dot product $a_1 \cdot a_3 \cdot b_1$. To do so, P_1 locally computes a dot product of a_1 and a_3 before running a secure $\mathsf{DotProd}$ with P_2 (see Step 3b). The function apriori-gen is for improving the computation – it helps to generate the set of *candidate* itemsets for L_k.

D Our Secure Dot Product Protocol

See Protocol 15.

Algorithm 1 `apriori-gen`(L_t)

```
Find all pairs of itemsets in $L_t$ where the first $t-1$ items are identical.
        e.g., $t = 5$ and two pairs $\{a, b, c\}, \{a, b, d\}$
Union them (lexicographically) to get a list of candidates $C'_{t+1}$
        e.g., $\{a, b, c\}, \{a, b, d\} \to \{a, b, c, d\}$
Prune $C_{t+1} = \{c \in C'_{t+1} \mid \forall s_c \notin L_t\}$, where $s_c$ is a $t$-subsets of $c$.
Return $C_{t+1}$
```

Figure 1: A Simplest `apriori-gen` Algorithm [3, 43]

PROTOCOL 15. (*Secure Dot Product* - $\Pi^{(t,n)}_{\mathsf{DotProduct}}$)

PARAMETERS:

- An upper-bound t.
- n parties: $P_1, \ldots, P_n$; an untrusted server $\mathcal{C}$;
- A PSI-CA functionality $\mathcal{F}_{\mathsf{PSI-CA}}$ in Functionality 4.
- A function $\mathbf{idx}' : \mathbb{Z}_2^\star \times \{0,1\}^\star \to (\{0,1\}^\star)^\star$ in Section 5.1

INPUTS:

- $P_{i\in[n]}$ has $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$.
- Cloud server $\mathcal{C}$ has no input.

PROTOCOL:

1. Each party $P_{i\in[n]}$ computes $A_i \leftarrow \mathbf{idx}'(X_i, t)$.
2. All parties invoke $\mathcal{F}_{\mathsf{PSI-CA}}$ where P_i inputs A_i, $\mathcal{C}$ inputs nothing, and P_1 obtains the output $|\bigcap_{i=1}^n A_i|$.

E Server-Aided 2-Party PSI Protocol[28]

See Protocol 16.

PROTOCOL 16. (*Server-Aided 2-Party PSI [28]*)

Parameters: There are 2 parties P_1, P_2 and a third-party server S. P_1 and P_2 have sets X_1 and X_2 as input, respectively. The server S does not have input. Let F be a PRF, and parameter $d > 0$.

Protocol:

1. P_1 chooses sets D_0, D_1, D_2 and a key k_1 such that $|D_0| = |D_1| = |D_2| = d$, sends them to P_2 and set $Y_1 \leftarrow X_1 \cup D_0 \cup D_1$.
2. P_2 sets $Y_2 \leftarrow X_2 \cup D_0 \cup D_2$.
3. P_2 chooses a random key k_2 and sends it to the server S.
4. Party P_1 sends a shuffled version of $Y_1' = \{F(k_1, x)\}_{x \in Y_i}$ to S.
5. The server returns a shuffled version π of $Y_1'' = \{F(k_2, y)\}_{y \in Y_1'}$ to P_1
6. Party P_2 sends a shuffled version of $Y_2'' = \{F(k_2, F(k_1, x))\}_{x \in Y_2}$ to P_1.
7. P_1 computes $I = Y_1'' \cap Y_2''$ and sends the result to P_2
8. P_2 computes $I^{-1} = \{F^{-1}(k_1, F^{-1}(k_2, x)) | \forall x \in I\}$
9. P_2 check that I has the right form and aborts if:
   (a) Either $D_0 \not\subset I^{-1}$ or $D_2 \cap I^{-1} \neq \emptyset$
   (b) There exists $x \in X_2$ and $\alpha, \beta \in [\lambda]$ such that $x||\alpha \in I^{-1}$ and $x||\beta \notin I^{-1}$
10. If P_2 does not abort, it notifies S who sends the shuffled function π to P_1. P_1 uses π learns the values in the set I^{-1}
11. P_1 checks that I has the right form as in Step (9) and aborts if the check fails.
12. The parties output distinct items in $I^{-1} \setminus D_0$.

F Zero Sharing Protocol [31]

See Protocol 17.

PROTOCOL 17. (*Zero-Sharing* - Π_{ZS} *[31]*)

Parameters: There are n parties $P_1, \ldots, P_n$. There is a PRF $F : \{0,1\}^\kappa \times \{0,1\}^\ell \rightarrow \{0,1\}^\kappa$.

Protocol:

1. Each party P_i picks a random seed $r_{i,j}$ for $j \in [i+1, n]$ and sends $r_{i,j}$ to P_j. The key K_i of party P_i is $(k_{1,i}, \ldots, k_{i-1,i}, k_{i,i+1}, \ldots, k_{i,n})$.
2. To obtain its share for value x, party P_i computes

$$S(K_i, x) = \left(\bigoplus_{j<i} F_{k_{j,i}}(x) \right) \oplus \left(\bigoplus_{j>i} F_{k_{i,j}}(x) \right)$$